THE UNIVERSITY OF CHICAGO

IDENTIFYING AND MITIGATING AI-ENHANCED PRIVACY ATTACKS

A DISSERTATION SUBMITTED TO
THE FACULTY OF THE DIVISION OF THE PHYSICAL SCIENCES
IN CANDIDACY FOR THE DEGREE OF
DOCTOR OF PHILOSOPHY

DEPARTMENT OF COMPUTER SCIENCE

BY
ZHUOLIN YANG

CHICAGO, ILLINOIS
JUNE 2025

To my parents, my lifelong friends, teachers, and greatest cheerleaders.

To my younger brother, my best buddy, who always brightens my day.

To my caring advisors, who guided me to be the researcher I am today.

I am deeply grateful to have you all in my life.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ABSTRACT

There is a long history in the security community of identifying privacy risks and attacks from the exposure of data to third parties. Over time, attacks have become more powerful and effective at extracting private data from ordinary observations. For much of this century, most privacy attacks on different data modalities and computing platforms have grown to rely on techniques driven by statistical machine learning (ML).

Nevertheless, this landscape is changing, with the arrival of significantly different AI/ML models and architectures capable of extracting patterns and information once considered out of reach of statistical ML models. Ordinary data once thought to be "squeezed" dry are now revealing entirely new and unexpected results through the application of these advanced AI/ML techniques. This in turn requires us to reassess and redefine what we consider possible in the world of privacy attacks.

To investigate this direction, this thesis primarily takes a traditional red team approach in the security domain. It first identifies new privacy attacks by integrating and adapting advanced AI/ML algorithms. It then discusses and proposes mitigation to address the newly identified threats. Specifically, the thesis discusses the process of fundamentally redesigning two well-known privacy attacks: keystroke inference and user interface confusion. The attack impacts are significantly enhanced by adapting advanced AI/ML techniques, such as self-supervised and self-attention learning, to the context of security.

The above findings highlight the urgent threat posed by enhanced attacks and emphasize the need to study new and robust defenses. Along this line, this thesis also explores a new authentication scheme to defend against AI-enhanced biometric spoofing (e.g., deepfake). The scheme employs carefully crafted autoencoder and convolutional architectures to capture subtle physiological features of human muscles that are difficult to replicate.

CHAPTER 1

INTRODUCTION

There is a long-standing history of identifying privacy risks associated with unwanted data exposure to adversarial parties, which persist alongside advancements in technology. In the nineteenth century, wiretapping emerged with the rise of telephones to eavesdrop on private conversations. In the twentieth century, brute-force attacks stealing passwords became prominent with the advent of computers, which automated the attack process. By the late twentieth century, phishing and man-in-the-middle attacks became widespread with the boom of the Internet, enabling attackers to manipulate communications. Throughout much of this century, data extraction attacks have attracted significant attention with the widespread adoption of data-driven system designs, where machine learning (ML) methods are employed to uncover complex patterns from observed data.

Most data extraction attacks leverage statistical ML algorithms to extract sensitive information from data. A well-known example is the keystroke inference attack. In 2005, security researchers demonstrated the feasibility of extracting keystroke information from the sound a user makes when typing on a mechanical keyboard [278]. This attack is made possible by statistical ML, which correlates keystrokes with acoustic patterns via statistical patterns found in languages. This is a practical attack in a quiet environment such as offices, which relies on just a single assumption: the attacker places a microphone nearby to record the typing sound. However, as people increasingly work in noisy, public environments, the effectiveness of this attack is significantly reduced.

Since then, many have proposed variations of keystroke inference attacks using different data modalities (e.g., WiFi [267] and reflections of keyboards [189, 258]) and computing platforms (mobile and virtual reality devices [266, 138]). However, these attacks rely on strong assumptions, such as access to labeled training data and prior knowledge about the target, to cope with increasingly complex attack environments. While these strong assumptions make the attacks feasible, they also significantly limit the real-world applicability of the attacks. Overall, while new forms of keystroke inference attacks continue to emerge, statistical ML remains the driving methodology

for achieving the most practical attack.

At this point, it seems that we security researchers have hit a wall in data extraction attacks, largely due to the use of statistical ML techniques to extract data patterns, i.e., we have "squeezed" the data dry. However, with the arrival of new AI/ML architectures, it is time to ask the question:

Can we break this wall?

This dissertation answers this question by adapting advanced AI/ML techniques, originally developed for benign applications, to the context of privacy attacks. New AI/ML models and architectures come with powerful interpretation, mimicry, and interpolation mechanisms. For instance, a well-trained attention model can extract very subtle vital signs, like heartbeats and respiratory patterns, from live videos of human faces [47]. As such, ordinary data once thought to be "squeezed" dry are now revealing entirely new and unexpected results through the application of these advanced AI/ML techniques. However, directly applying these advanced techniques to privacy attacks is impractical, as they often require a substantial amount of labeled training data—an assumption that is challenging if not infeasible for attackers. Therefore, this dissertation focuses on exploring whether these advanced AI/ML techniques can be adapted to enable unexpectedly powerful attacks under practical attack assumptions.

With these in mind, this dissertation explores three directions: (1) It redesigns classic privacy attacks by adapting advanced AI/ML techniques. In particular, it studies AI-enhanced keystroke inference attacks to showcase the profound impact. (2) It identifies new privacy attacks by enabled by advanced AI/ML techniques. Particularly, it studies a new interaction manipulation attack in virtual reality environments, whose potential harms are significantly amplified by the use of generative AI. The findings in (1) and (2) stress the threats posed by AI-enhanced privacy attacks and underscore the urgent need for adaptive defenses. (3) In line with this, the dissertation designs a user authentication system to defend against AI-enhanced biometric spoofing.

1.1 Redesigning Keystroke Inference Attacks

Users frequently enter sensitive and confidential information into computing systems by typing on keyboards, making keystroke inference attacks a classic and well-recognized threat. Over the past decades, a number of keystroke inference attacks have been proposed, each relying on specific assumptions [278, 258, 189, 266, 138, 122], such as access to labeled data from the target user or the presence of nearby sensors (like microphones). Yet a practical keystroke inference method that operates effectively under general assumptions remains an open challenge.

This dissertation redesigns keystroke inference attacks by adapting new AI/ML techniques (§2). It proposes a general attack, where the attacker does not rely on any prior knowledge or labeled data on the target user or their device. By analyzing a short video of the target's typing taken meters away, the attacker can extract over 90% of the typed content (see Figure 1.1). The core of this attack is a self-supervised learning pipeline, which curates labeled data from the video and trains deep learning models to achieve robust inference results.

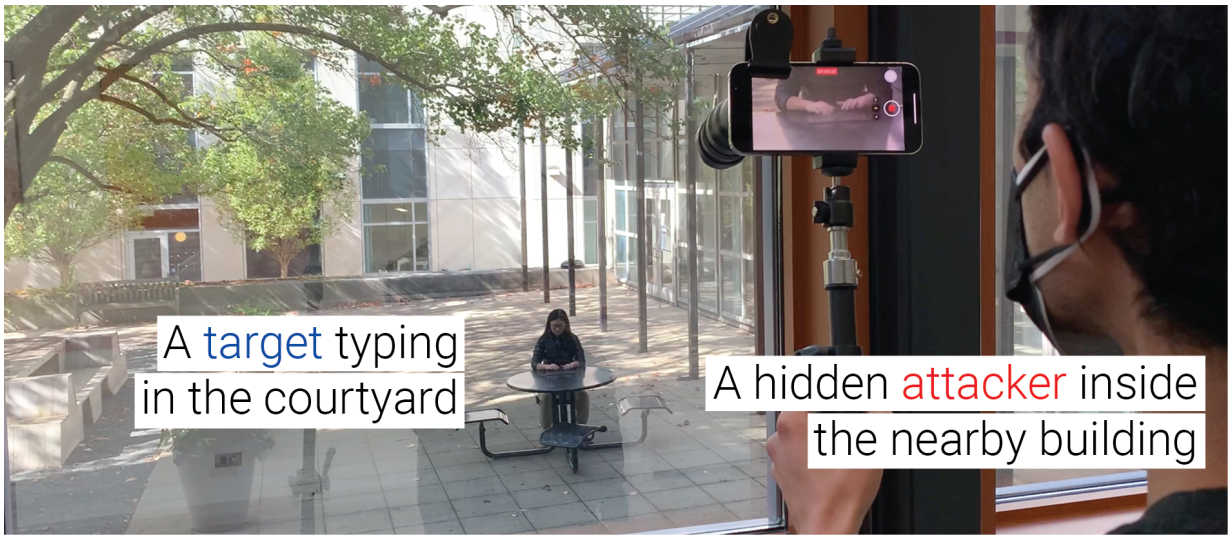

Figure 1.1: A long-range keystroke inference attack where the attacker hides inside a building (behind a window) using a smartphone to record the target in the courtyard (≈12m away) typing. The recorded video is the only thing the attacker needs to infer keystrokes.

The self-supervised learning is built on two general principles in human typing. The first is a mechanical principle, where a fingertip must press on the keyboard to input a key. The pipeline

relies on identifying these pressing moments to label video frames with keystroke events. The second is a semantic principle, where keystrokes have underlying statistical relationships guided by the language the target user types. The pipeline relies on identifying these underlying relationships to learn pseudo labels of the keystrokes. Finally, the pipeline utilizes the pseudo labels to train a deep learning based keystroke classifier for inferring the typed content from the entire video.

The dissertation applies this self-supervised learning methodology to build two potent keystroke inference attacks, one in the physical world (§2.1) and one in virtual reality (VR) (§2.2). As shown in Figure 1.1, an attacker can infer the target's typed content from meters away by recording a 10-minute video of the target's typing hands. Similarly, in a shared VR work environment, the attacker can successfully recover the typed content by screen-recording the target's avatar for 10 minutes. This is made possible by the addition of self-attention models designed to mitigate digital noise in VR motion capture. This dissertation discovers that complex self-attention models can be effectively trained on limited data when given a simple learning goal. Based on this insight, self-attention models are successfully trained even in data-scarce attack scenarios.

1.2 Hijacking Communications in Virtual Reality

VR is an immersive interface that transports users to lifelike digital environments. Within these environments, users are represented by realistic avatars—digital selves that mirror facial expressions and body movements. These avatars enable natural user-to-user communication, allowing users to interact with others much as they do in physical settings such as offices, courtrooms, and recreational places like national parks. Although VR is still emerging, substantial research and a growing array of products and applications have already been published. For instances, Meta has developed Pixel Codec avatars achieving photorealistic user representation [158]; BRINK Traveler app provides photorealistic natural wonders scanned in the physical world for users to visit together in VR [257]; Judge Andrew Siegel used VR to step into the crime scene and better understand a defendant's testimony [185].

VR significantly advances one key aspect of modern interfaces: user-to-user communication. Text messages, phone calls, and video conferencing tools like Zoom fall short of replicating the natural flow of in-person conversations. This gap exists because key nonverbal elements such as body language, shared spaces and objects, social proximity, and even attire are often absent. Notably, these missing elements can all be reintroduced through the immersive and embodied experiences that VR uniquely enables (see Figure 1.2).

Figure 1.2: A future VR workspace: Betty puts on a VR headset and enters a virtual workspace where she meets John. What she sees is John's avatar which reflects not just his appearance, but also real-time facial expressions and natural behaviors with striking realism. Credit: Nimito.

At this point, one key security question arises: How might privacy attacks evolve in this new immersive interface? One set of well-known privacy attacks targeting computing interfaces is the user interface (UI) confusion attacks [31]. They often happen on web browsers and mobile devices. They overlay legitimate UI components (e.g., text boxes and buttons) with malicious but visually identical ones to trick users into leaking private information such as passwords. However, UI confusion attacks ought to be very different inside VR. Specifically, they should confuse the communications between users, effectively compromising the relationship between parties.

This dissertation takes an initial step to study the above security question. In particular, it designs a new form of UI confusion attacks, namely interaction manipulation attack, where a remote attacker takes control of a user's interaction with their VR system by trapping the user inside a

malicious app that masquerades as the full VR interface (§3). Once trapped, all of the user's interactions with apps, services, and communications with other users can be recorded and modified without their knowledge. The attack is implemented on Meta Quest VR headset, one of the most popular VR platforms. The feasibility, efficacy, and stealthiness of the attack are extensively evaluated with multiple user studies. Finally, this dissertation emphasizes that emerging generative AI techniques will significantly enhance the impact of this attack, such as those capable of replicating human voices or generating deepfake videos in near real-time.

1.3 Defending against AI-enhanced Biometric Spoofing

The above attack findings highlight the urgent threat posed by AI-enhanced attacks and emphasize the need to study more robust defenses. Along this line, this dissertation explores and proposes a new authentication system (§4) to defend against AI-enhanced biometric spoofing (e.g., deepfake faces and voices). The system employs carefully crafted AI/ML architectures to capture intricate physiological features of human muscles that are difficult to replicate by today's AI/ML methods.

In particular, the system utilizes electrical muscle stimulation (EMS), which has been extensively studied in the Human-computer Interaction community. It is known for its intersubject variability, i.e., different individuals respond differently to the same EMS stimulation because everybody's physiology is different. Another unique feature of EMS responses is its multiplex intrasubject variability. Each individual responds differently to different stimulation. More importantly, EMS stimulation varies in location, coverage, intensity, duration, and frequency, creating nonlinear effects on EMS responses that are difficult for current generative AI techniques to replicate (compared to biometrics like face, fingerprint, and voice).

The authentication system adapts a two-step approach in response to the above two features. First, we employ an autoencoder-based anomaly detection to extract the key representation of a user's responses. This design amplifies and quantifies the difference between others' and the target user's responses, effectively defending impersonation attacks. Once we confirm that the response

originates from the target user, we use a deep learning classifier to identify the corresponding stimulation. If the identified stimulation matches the one currently issued by the system, the authentication is successful. This latter step is designed to defend against replay attacks, where an attacker attempts to bypass the system by replaying a previously captured legitimate response. In addition, the system is capable of generating millions of stimulations due to the multiplex intrasubject variability. As a result, each stimulation-response pair can be discarded after use, effectively making each pair a one-time biometric password.

1.4 Content Summary and Outline

This dissertation highlights the importance of investigating AI-enhanced privacy attacks and its contributions in §1. It then explores three directions: (1) Redesigning classic privacy attacks, (2) Identifying new privacy attacks, and (3) Defending against AI-enhanced attacks. Specifically, it first redesigns a classic privacy attack—keystroke inference—by adapting self-supervised learning and self-attention techniques (§2). It then explores the emerging VR interface and identifies a new privacy attack, i.e., interaction manipulation attack, highlighting the potential harms posed by generative AI (§3). Finally, it defends against AI-enhanced biometric spoofing by proposing a user authentication system that leverages unique human muscle involvement (§4).

CHAPTER 2

REDESIGNING KEYSTROKE INFERENCE ATTACKS WITH SELF-SUPERVISED LEARNING

2.1 Towards a General Video-based Keystroke Inference Attack

2.1.1 Introduction

Jane walks into an airport lounge. With her flight boarding in an hour, she has just enough time to get online to write a few emails and pay some bills. Jane has heard stories about privacy risks of working in public places, e.g. people reading over shoulders and even stealing passwords with recording devices. So she finds an empty table in a corner, and before sitting down, checks the nearby area (e.g. ceiling, under the table) for sensing devices. Satisfied, she pulls out her iPad (with a privacy screen cover), and starts working. As Jane watches some passengers walk by and a few others sitting with their own mobile devices, she wonders: "Is it safe for me to write sensitive content/emails or type passwords? Have I taken enough precautions to protect myself?"

In the age of machine learning and remote work, Jane's concerns are actually quite realistic (and common). Machine learning tools have grown increasingly proficient at extracting keyboard keystrokes from a variety of side channels and sensory data. Meanwhile, accommodations for remote work have untethered employees from their offices, and work is often done on the road or in public settings, e.g. airports, coffee shops, trains and airplanes. For millions of affected workers, a simple question remains: "Are reasonable precautions enough to protect them and their data from invasive keystroke inference attacks in real-world settings?"

Despite extensive prior work on keystroke inference attacks, the answer to this question remains unclear. This is due in part to the reliance of existing attacks on novel but restrictive scenarios where the attacker has access to specific types of sensor data or side-channel information.

We consider existing keyboard inference attacks in two broad categories: vision-based attacks,

and non-vision attacks (e.g. everything else). The latter group does not rely on vision techniques, but instead distinguishes keystrokes using data (e.g. audio, vibration) gained by placing sensors close to Jane. For example, audio-based attacks place a microphone next to Jane to capture key-specific sounds generated from typing on a mechanical keyboard [278]. RF-based attacks [120, 267] place WiFi devices close to Jane (e.g. 30cm) to capture subtle signal variations caused by her one-finger key entries. Other work explores the use of electromagnetic (EM) or LTE measurements to infer one-finger key entries. To succeed, they require either placing an EM sniffer right under Jane's table [104], or zero movement anywhere within 20 meters of Jane [125].

Figure 2.1: Sample attack scenarios: (a) an indoor lounge scenario where the attacker watches a video while recording the victim typing; (b) a long-range outdoor scenario where the attacker hides a smartphone with a budget telephoto lens (<$60) inside a building (behind a window) to record the victim in the courtyard (≈12m away) typing; (c) the victim can type on a physical keyboard (here, an example of iPad keyboard) or even (d) uses an "invisible" keyboard and types directly on the table.

The other category of attacks is vision-based, and generally rely on strong assumptions on specific viewing angles, or other extra information such as precise keyboard layouts and reflective keyboard surfaces. Some attacks rely on direct (or birds-eye) views of Jane's keyboard and fingers, by either placing a camera above Jane [27] or by capturing a view of the screen reflected by her eyeballs [189, 258]. To help train inference models, a recent work produces synthetic data by overlaying a thumb image on mobile keyboards [122]. In contrast, other attacks operate on "normal" frontal views, but require extra visual cues to locate Jane's pressing fingertip. Not only must attackers know the exact keyboard location/size/layout, they also need added info such as reflections around the pressing fingertip, created by a reflective typing surface [265], or the ground truth location of a key in both the video and the typed content, i.e., the "Enter" key on a PIN pad [210].

We note that a privacy-aware user can effectively disable most, if not all of these attacks. For example, Jane can avoid RF/EM attacks by looking around her work area for sensing devices, while the complex motion of her touch typing (using 10 fingers) is much harder to distinguish via RF/EM signals compared to 1-finger typing targeted by prior attacks. She can protect herself against existing vision-based attacks by checking for overhead cameras, and her eyes are naturally protected just by looking down at her device. Matte screens or screencovers will disable fingertip reflections while touchscreens provide reconfigurable keyboard layouts. Finally, audio based attacks are ineffective on today's touchscreen keyboards.

A general vision-based keystroke inference attack. In this paper, we want to understand if today's computing users are vulnerable to keystroke inference attacks in general settings, after taking simple precautions such as those mentioned above. We ask if it is possible to recover text typing, by simply pointing a single RGB camera at a user's hands from a distance. Without external information from side-channels, can attackers invade a user's privacy in realistic settings such as airports, coffee shops, or outdoor courtyards? These represent the typical mobile typing settings for users on-the-go.

In this more general threat model, the attacker has no information about Jane, except that she types in English. The attacker has no knowledge of Jane's keyboard location, size, or layout, no videos of Jane typing known text, no knowledge or use of any visual cues, no access to or control of any sensor, device, and side-channel beyond a single RGB camera observing Jane at a distance. This threat model is designed to realistically capture what an attacker can do on a first encounter with a target. Figure 2.1 illustrates several sample scenarios covered by our threat model.

Keyboard inference in this general threat model is extremely challenging for several reasons. First, human typing is a complex process that is user-specific, highly variable, and heavily dependent on content and keyboard structure/layout [28, 59, 42]. Thus it is impractical to train a keystroke classifier that generalizes to different targets, devices and environments. Second, observations in a realistic attack are short, e.g. 15-minutes captures $\sim$3000 keystrokes, 2 orders of

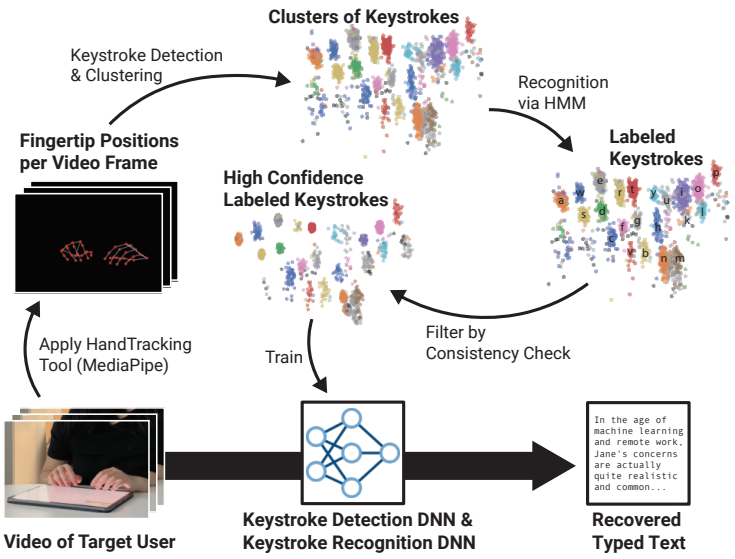

Figure 2.2: Our self-supervised approach to keystroke inference. We first run unsupervised inference on fingertip data extracted from each video frame, from which we identify keystrokes with high confidence labels (this process is marked by thin arrows). We use these as training data and build DNN models that detect and recognize keystrokes directly from the video (thick arrow).

magnitude less than required for general self-supervised vision tools [61, 22]. Finally, hand tracking tools try to identify keypress events in videos, but suffer from large tracking errors due to artifacts in RGB video like depth ambiguity, finger occlusion[1] and motion jitter.

A self-supervised approach to keystroke inference. In this work, we propose a new approach to keystroke inference with no additional input other than video captured from a distance via commodity phone cameras. The key insight is to use a two-layer self-supervised system, where noisy results of hand tracking on the target video are used to run keystroke detection/clustering, followed by a language-based Hidden Markov Model (HMM) to recognize keystrokes. These initial labels are filtered using multiple consistency checks to produce high confidence labels on video frames, which are then used to train two 3D-CNN models that detect and recognize keystrokes from the video. This two-layer process is illustrated by Figure 4.1.

We evaluate this attack using IRB-approved user studies under a variety of conditions, vary-

1. In frontal views, some fingers can be blocked by the fingers in front of them just enough that hand tracking cannot properly track them.

ing the target (user/typing behavior, keyboard device, content typed, physical environment) and attacker behaviors (hand tracking tool, attack distance). The attack is highly effective in nearly all settings, and performs well across our user study participants, despite significantly different typing styles and abilities.

Ethics. Our goal is to bring attention to privacy risks from video-based keystroke inference attack in public settings. Beyond careful user study design to minimize participant harm, we believe our study can increase awareness to protect users, and lead to further adoption of simple and effective mitigation awareness, e.g. portable barriers to prevent line-of-sight.

2.1.2 Background and Related Work

We begin by discussing human typing, existing keystroke inference attacks, and vision based hand tracking.

First, typing is a cognitively complex process that relies on many human factors: language/memory faculties, attention states, and typing muscle memory [103]. Human typing behaviors are not only complex, user-specific, and time-varying, but also heavily dependent on content, keyboard/input device and other environmental factors [28, 37, 59, 42, 74]. Different typing styles and motions are a main reason why there are no known models that reliably extract multi-finger keystrokes from human hand/finger behaviors.

2.1.2.1 Existing Keystroke Inference Attacks

There are numerous keystroke inference attacks in existing literature. Given our focus on general keystroke inference, we *do not* consider special subcases such as "invasive" attacks where the attacker has internal control over the user's device, and actively controls and/or listens to its sensors [136, 250].

We broadly categorize existing attacks into two categories: non-vision attacks and vision-based attacks.

Non-vision attacks. The attacker collects data about the target's typing by placing sensors near them. Sensors might include audio microphones or side-channels such as vibration, electromagnetic (EM) and RF.

Audio. Acoustic-based attacks place a microphone next to the target's keyboard to capture key-specific sounds generated by a mechanical keyboard [278], and its attack range can be extended to 15m using a bulky parabolic microphone [23]. These attacks work on multi-finger typing, but depend heavily on keyboard sound quality. They are generally ineffective on today's touchscreen keyboards which produce little sound.

Vibration. Keypress events generate subtle vibrations. An accelerometer within 5cm of a keyboard can pick up vibrations induced by typing [143]. The physical proximity required by this attack makes it easy to detect in practice.

EM. Recent work identified that typing on a touchscreen generates EM signals (via coupling), which vary with keys [104]. This attack only supports 1-finger inputs on a PIN pad, and requires an EM sniffer placed right under Jane's table. The attacker must know the PIN pad layout and position.

RF. By placing WiFi devices close to the target (e.g. 20cm [267]), an attacker could capture the subtle WiFi signal variation caused by 1-finger key entries. One attack [120] achieves 1.5m attack distance but requires the target to connect to an AP that the attacker controls. Furthermore, these attacks all require the exact PIN pad layout/position and user-specific training data [267, 45, 120]. Another study [10] targets multi-finger typing, but again requires placing WiFi devices close to Jane (30cm) and user-specific supervised training.

Cellular LTE signals can also be leveraged to infer 1-finger typing. A software defined radio within 15m of the target PIN pad can capture the LTE signal (sent by a LTE base station within 150m) reflected by Jane [125]. This attack fails if there is any moderate movement anywhere within 20 meters of Jane. Again, this attack requires knowledge of the keyboard layout and user-specific training.

Vision-based attacks. Vision-based attacks also often target 1-finger typing. We divide them into two groups based on the angle or "view" of the attacker.

Birds-eye view. Many attacks require a direct (bird's eye) view of the target's keyboard and fingers, as if the attacker is viewing through the target's eyes. This is done by either placing a camera above (or just behind) the target, depending on how the target places or holds the keyboard [27, 122], or by capturing the screen reflected by their eyeballs[2] [189, 258].

Frontal view. Other attacks can use indirect "frontal" views, but require extra visual cues to locate fingertip keypresses. Aside from knowing the exact keyboard location/size/layout, the attacker must know the lighting/reflection patterns around the fingertip (by relying on a reflective typing surface [265, 266]), or the precise location of a specific (frequently used) key in both record video and the typed content, i.e., the "Entry" key on a PIN pad [210]. Finally, it is possible to record the target's upper body movement when typing, e.g. during a video chat, and use them to infer keystrokes [198]. Again, the attacker must know the exact keyboard layout.

Summary. Existing work has demonstrated the feasibility of keystroke inference attacks under novel but often restrictive scenarios where the attacker has access to specific types of sensor data and/or keyboard information. In this work, we consider a general (and more realistic) attack scenario, where the attacker uses only a frontal view of Jane's typing hands and nothing else (see the threat model in §3.3.1).

2.1.2.2 Vision-based Hand Tracking

Given our goal of general keystroke inference without side-channel data, we have to incorporate current vision tools for hand tracking. 3D hand tracking (or handpose estimation) is a long-standing problem in computer vision, and today's tools provide good but still noisy results. We describe current tools here, and later discuss how our system design overcomes errors generated by tools

2. The target holds a phone vertically while thumb typing. Thus their eyeballs or sunglasses reflect the phone's screen and the typing finger.

such as MediaPipe.

There are two types of hand tracking tools available today. Depth-based hand tracking [44] requires a high-precision depth sensor, which are either bulky (e.g. Microsoft Kinetic) or limited to short distance (e.g. iPhone's depth sensor works within a range of 50cm). In contrast, RGB-based hand tracking supports longer range, but is much more challenging due to occlusion, depth ambiguity, significant variation in camera viewpoint and appearance condition [162]. Today's SOTA models provide cm-level accuracy on known poses [115, 251, 268, 119] and 5cm mean error on others [162]. To our knowledge, there is no specialized hand tracking tool for keystroke detection. Prior work on keystrokes [74] tracks fingers by placing 52 reflective markers on hands and using 8 infrared cameras recording at 240fps, far from our realistic attack scenarios.

MediaPipe. Several general-purpose hand tracking tools can extract arbitrary handposes from RGB videos at 30-60fps, and the most well-known is MediaPipe [269, 237] (Google 2020). It extracts handposes as 2.5D coordinates of 21 joints per hand (horizontal, vertical, depth relative to the wrist). The public release includes only the binary code, without the DNN model or its training data. Our work uses MediaPipe to extract handposes from recorded typing videos.

2.1.3 Threat Model

In pursuit of a realistic attack in common everyday settings, we consider users who might be vulnerable while working in public places like cafes, airport lounges, or outdoors in courtyards or on park benches. These users use *mobile typing*, i.e., typing on iPads or portable keyboards that are designed for computing users on-the-go. We consider an attacker who records them typing (with a frontal-view video of their hands) from a distance using a single RGB camera (e.g. a commodity camera phone), then processes the video to reconstruct the typed content. Thus, we make four simple assumptions:

- The attacker knows the language used by the target (English in this work).

- The attacker has a frontal view of the target's hands.
- The attacker's camera remains stationary (placed on a stand in this work).
- The attacker has access to a hand tracking tool that operates on the frontal-view video (MediaPipe in this work).

Our work differs significantly from prior work in that we do not rely on side-channel data or other assumptions. These are assumptions that we **do not make**:

- The attacker has no knowledge of the target's keyboard layout (keyboard could be customized via third party apps like Gboard [18]), and may not have a clear view of the keyboard (e.g. typing on an iPad with a screen protector or a projection-based virtual keyboard)
- The attacker has no labeled data or prior observations of the target. This follows our assumption that the attack is opportunistic and has no prior planning.
- The attacker cannot install, access and manipulate any sensor, device and channel beyond the single RGB camera at a distance from the target.

2.1.4 Design Alternatives and System Overview

In this section, we consider the challenges of a general keystroke inference attack, weigh two potential solutions, and describe their shortcomings. We then present a new self-supervised approach which when given a video, extracts specific frames and labeled keystrokes, and uses them to train customized DNN models that accurately detect and recognize keystrokes for that video.

2.1.4.1 Potential Solutions and Their Limitations

In exploring the feasibility of a general keystroke attack, we considered a number of possible approaches, all of which produced less than satisfactory results. We found two challenges to be the most difficult to overcome. First, users consistently hit edges of keys as they typed, meaning

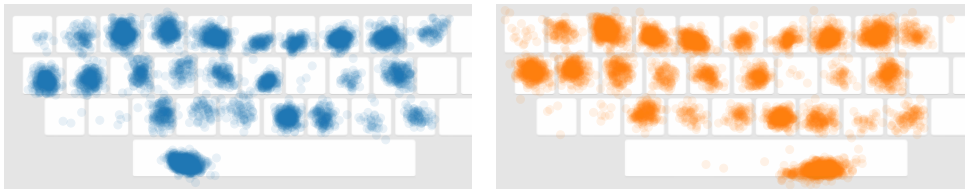

Figure 2.3: Touchpoints of keystrokes recorded by a touchscreen keyboard, where each circle is a touch point. The separation between neighboring keys' touchpoints is barely $1cm$ in average.

that for most users, the separation between positions of their fingers hitting two neighboring keys, is quite small. Figure 2.3 confirms this, using actual recorded positions of how two different users typed on their iPad keyboards. Second, we found that a reasonable length video (e.g. 10mins), provided roughly 3000 keystrokes for moderate speed typists, which is insufficient data for existing self-supervised tools to train a classifier for 27 keys (26 letters and space bar).

Here we consider in detail two potential attack designs: (1) training a supervised DNN keystroke inference model on one set of users, and relying on model transferability to successfully apply that model to videos of other target users; (2) using unsupervised inference based on handpose data extracted from the video (using hand tracking tools), which is easier to model and interpret compared to a DNN. As we discuss below, we find that both faced significant limitations under our attack scenario.

Transferability-based attacks. An intuitive approach to general keystroke inference is to perform supervised training of a DNN keystroke inference model using labeled data collected from a set of users, then apply it to other target users. This leverages the concept of model transferability, the idea that models trained for one instance of a task can perform reasonably well on other instances of that or similar tasks.

In practice, we find that supervised inference models trained on one set of users fail to generalize when applied to videos of other users. The implication is that the mapping from movements to keystrokes is user-specific enough to prevent transferability across users.

To confirm this, we recorded videos of 16 users typing on the exact same keyboard with the same camera angles. We applied transfer learning (using a well-known gesture recognition DNN

model) to train keystroke recognition models, and performed leave-one-out cross validation. In each experiment we used labeled data from 15 users to train and tested the model on the 1 left-out user. While the trained models can correctly decode 99+% of keystrokes from any trained user, the transferability to the new user is very low – the mean character error rate is 48% and the word error rate is 98% across all the experiments. Note that this is already under near-optimal conditions where everyone uses the **exact same** keyboard.

Unsupervised inference using fingertip data. Without labeled training data of the target, one practical alternative is to run unsupervised inference directly on a processed version of the visual data. Since keystrokes directly result from fingertips touching the keyboard, the attacker can apply a hand tracking tool on the video to extract, for each frame, the fingertip locations of the ten fingers, and use this data to detect and distinguish between different keystrokes. Such analysis can then be combined with language-based models like HMM to infer the key of each detected keystroke. This is similar to the methodology used by prior audio-based attacks [278].

While attractive, this solution relies heavily on the accuracy of the fingertip data extracted from the RGB video. Since neighboring keypresses are separated on average by less than $1cm$ (see Figure 2.3), the finger tracking precision needs to be at the level of millimeters. This is unfortunately infeasible with today's hand tracking tools. The resulting errors in the fingertip data propagate into the inference pipeline, and significantly degrade inference accuracy. Later in §2.1.5.5 we present a detailed study to illustrate its impact using different hand tracking tools.

Manual attacks. We also attempted a manual attack by studying the recorded video frame-by-frame and labeling them with the observed keypresses, if any. While seemingly easy, identifying/locating a keypress is quite difficult for human eyes. The combination of depth-ambiguity, finger-occlusion, and acute viewing (i.e., frontal rather than top-down view) makes multiple fingers appear equally "close" to the keyboard. As such, many keypresses were either mislabeled or not detected at all.

2.1.4.2 Key Insights

Curating labeled training data for a target video from its own fingertip inference result. When we observed that unsupervised inference on fingertip data is highly susceptible to hand tracking errors, we noted that its inference result is quite skewed: some keystrokes are accurately predicted while others are erroneous and cannot be corrected using post-analysis tools. If we can identify these accurately predicted keystrokes, we can use them as labeled training data to train DNN models that accurately detect and recognize keystrokes on the **same** video. Since the DNN models operate on raw video frames, they are no longer affected by errors introduced by hand tracking tools.

Identifying high confidence labels using consistency checks. We propose to identify correctly predicted keystrokes by checking the consistency between a keystroke's inference result (produced by a language model) and its spatial position on the keyboard estimated from its fingertip data. For all keystrokes assigned for the same key, the points where they touch the keyboard should form a tight cluster.

2.1.4.3 Attack Design: Overview

Following the above insights, we propose a new attack approach, which applies two layers of data analysis on an attack video to curate labeled training data and then train DNN models that detect and recognize the keystrokes from the same video (see Figure 4.1). These high-performance DNN models take as inputs a sequence of raw video frames (rather than their hand tracking results) and output a sequence of characters as the recovered content.

Overall, the attack includes the following two steps, one per layer. We discuss each in detail in the subsequent sections.

Step 1: Unsupervised inference on handpose data (§2.1.5). Given a video on the target, we first apply a hand track tool (MediaPipe in our current implementation) to extract handpose

data from each video frame. We then apply an unsupervised inference pipeline on this sequence of (noisy) handpose data to detect and cluster keystrokes, followed by an HMM-based language model to infer the character of each detected keystroke. This creates an initial label for each keystroke frame of the video.

Step 2: Self-supervised DNN inference on video data (§2.1.6). Given the initial noisy label of the detected keystrokes, we apply consistency checks to identify keystrokes with high confidence labels, and use them to curate labeled training data to train a DNN-based detector of keystrokes and a DNN-based classifier to recognize the detected keystrokes (i.e., mapping each to a key). We apply multiple noise-aware training methods to address any residue errors in the curated training data.

2.1.5 Unsupervised Inference on Handpose Data

We start from the initial step of unsupervised inference on handpose data. This is done using a sequential pipeline: first detecting keystrokes (i.e, when a key is pressed), clustering keystrokes by their touchpoints, and applying a language-based analysis to estimate the typed content. While the method is similar to that of audio-based attacks [278], our contribution is realizing it in the context of general vision-based attacks. In the following, we describe the handpose data used by our pipeline, the three inference components, followed by a study on the impact of hand tracking noise.

2.1.5.1 Handpose Data

Our pipeline operates on the fingertip coordinates per video frame, identified by the hand tracking tool. This configuration is carefully chosen to address finger occlusion and depth ambiguity of the keystroke video.

Camera configuration. To extract handpose data, the camera needs to be positioned such that both hands are visible and that the hand tracking tool is working properly (i.e., no frequent flutter,

misaligned handposes). When using MediaPipe in our attack, we find that the camera-to-keyboard angle needs to be determined per target. To do so, we build an online calibration module leveraging the real-time hand tracking API provided by MediaPipe [147]. For a given camera position, this module inspects the temporal alignment of the extracted handposes over 15 seconds (by computing the cosine similarity between handposes across video frames), and if the handposes are sufficiently aligned, the camera position is suitable for the attack. In general, we find that the camera needs to be 10° above the keyboard and the calibration is quick (e.g. 15-20 seconds) for an experienced attacker.

2D fingertip data. We focus on fingertips rather than all 21 joints provided by MediaPipe [237], because a keystroke is produced by a fingertip pressing down on the surface. Figure 2.4 shows an example of MediaPipe's hand tracking on video frame. Inference using fingertip data incurs less complexity but also less tracking errors. Furthermore, while MediaPipe provides a 2.5D[3] coordinate per fingertip (i.e., the pixel coordinate x, y and a relative depth to the wrist), we find that the relative depth carries little information but much unwanted noise as the wrist moves naturally during typing. As such, we only use the 2D fingertip coordinates per video frame.

Non-thumb data only. In frontal views, it is difficult to capture all 10 fingers due to finger-on-finger occlusion, especially when typing with multiple fingers per hand. When using MediaPipe in real-world attacks, we find that the target's thumbs are often blocked by other fingers just enough to prevent MediaPipe from tracking them properly (e.g. the detected thumbs flutter frequently). Thus, we choose to operate on the 8 non-thumb fingertip data. Our design can still detect and recognize thumb-based keystrokes using non-thumb data, by leveraging natural correlation in finger motions.

Preprocessing. After extracting fingertip data from each video frame, we perform smoothing to remove potential noise. Here each fingertip has a sequence of pixel coordinate (x, y), one per video frame. We apply a standard low-pass butterworth filter with a cut-off frequency to smooth each

3. To the best of our knowledge, there is no tool providing 3D tracking of keystroking fingers in a frontal-view RGB video.

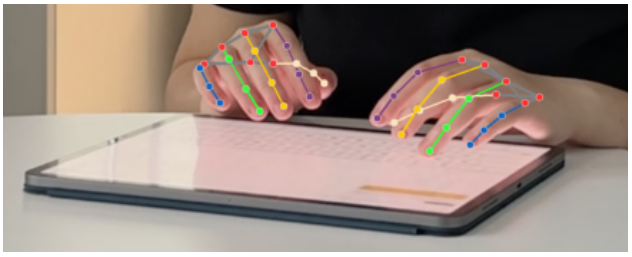

Figure 2.4: An example of MediaPipe hand tracking output.

individual fingertip's sequence. Since the common typing speed is around 200-300 keystrokes per minute (3-5Hz), we set the cut-off frequency to 6Hz.

2.1.5.2 Detecting Keystroke Events

Since the attacker has no knowledge (or even visual) of the keyboard, we propose to detect a keypress by detecting negative peaks on the fingertip acceleration – when a finger actively presses down and hits a key, its motion reduces/stops abruptly. Not yet knowing which finger touched the keyboard, we use the maximum value of the fingertip acceleration of all four non-thumb fingers (per hand) to run the detection. Here the acceleration is computed by taking the double derivative on each fingertip's y-coordinate across frames.

Handling spurious peaks. Interestingly, *not every negative acceleration peak maps to a keypress.* The spurious peaks come from two main sources: (1) the noise in fingertip data, and (2) the noise in human typing behavior since we often make unconscious hand movements similar to those of subtle keystrokes, e.g. when hesitating or thinking about what to type. As most spurious peaks have small prominence values, we apply statistical thresholding to filter them out. Rather than pre-defining a "magic" threshold, we compute a threshold for the current attack video by modeling the peak prominence[4] value p as a Gaussian mixture of keypress and no keypress ones. In this case, the dip between the two hills in the probability distribution of p would approximate the threshold required to produce equal misdetection and false alarm rates.

4. Peak prominence measures how much a peak stands out from the surrounding signal baseline, and can be computed via a SciPy function [53].

Can thumb-based keystrokes get detected? Using only non-thumb fingertip data, our design can still detect keystrokes made by thumbs. This is because the muscles used to move our fingers are inter-connected, and thus the finger movements naturally correlated. When we press a key using a thumb, the other 4 fingers on the same hand also move down with it. Our acceleration based detection can detect thumb-based keystrokes, often at an accuracy comparable to those of non-thumb keystrokes.

One would think that since the acceleration of thumb-based keystrokes is computed from non-thumb fingertips, it should be weaker than those of non-thumb keystrokes. It is not true according to our measurements – the two show similar average peak prominence, and some non-thumb keystrokes are weaker than thumb ones (see the peak prominence distribution in Figure 2.7 in Appendix). Thus in §2.1.5.3, we apply a different method to separate thumb and non-thumb keystrokes.

2.1.5.3 Clustering Detected Keystrokes

Next, we organize the detected keystrokes (a mix of thumb and non-thumb ones) into clusters. This clustering result is later used in conjunction with a language model to infer the typed content (§2.1.5.4). We cluster keystrokes by estimating their touchpoints on the target's keyboard, which directly relate to the typed key. The exception is thumb-based keystrokes, where we can only estimate the "fake" touchpoint made by a non-thumb finger. Thus, we propose to process them separately from the non-thumb keystrokes. With this in mind, our clustering process includes four steps: (i) identify the pressing fingertip, (ii) apply perspective transformation to convert a 2D fingertip coordinate (obtained via the frontal view) into a touchpoint on the keyboard (i.e., the birds' eye view), (iii) separate the detected keystrokes into 2 groups: non-thumb and thumb based keystrokes and finally, (iv) cluster keystrokes in each group based on their estimated touchpoint locations.

Identifying the pressing fingertip. Since finger movements are correlated [263], negative acceleration used in §2.1.5.2 can effectively identify the keystroking hand, but not the pressing finger. Instead, for each frame, we estimate the vertical displacement of each non-thumb fingertip from its average vertical location across the video, and locate the finger with the largest displacement (to reach the keyboard). We note that due to depth ambiguity, this identification method is more effective for keys in the front row since their displacement estimation is more accurate.

Estimating a keystroke's touchpoint on the keyboard via perspective transformation. The pressing fingertip's 2D coordinate (x, y) is from the frontal-view video frame, and thus a skewed/-compressed representation of its touchpoint on the target's keyboard. To reduce the effect of skew/-compression, we apply perspective transformation to map each (x, y) to a touchpoint on the target's keyboard (in a birds' eye view). We first mark the 4 points on the video to indicate the keyboard's planar surface[5]. We then compute a homography matrix H between this planar surface and the video frame's perspective, using an OpenCV function (`perspectiveTransform`) [65]. By multiplying (x, y) with H, we estimate its corresponding touchpoint on the keyboard.

Separating non-thumb and thumb keystrokes. We separate them by analyzing the standard deviation of the typing hand's 4 non-thumb fingers' displacements. This is because a thumb keypress would trigger similar motions at the 4 non-thumb fingers (moving down together). In short, their displacements would be similar, thus the stds are generally smaller than those of non-thumb keystrokes. We compute the threshold by treating the thumb keystrokes as a single key input, whose frequency is bounded by 20% [144]. This detection will introduce errors that depend on the target's typing behavior and keyboard layout.

Clustering non-thumb keystrokes. Given the estimated touchpoints of all non-thumb keystrokes, we run K-Means with 33 clusters to cover keys that can be inferred by a language model and to allow frequently used keys to form multiple clusters. This is because studies have shown that high

5. It could be 4 corners of the keyboard if the keyboard device is visible or 4 points on the table to indicate the planar surface.

frequency English keys can have more than 5 times amount of samples compared to low-frequency keys [236].

Clustering thumb keystrokes. We choose to cluster thumb keystrokes (instead of just separating them by the typing finger) to help mitigate errors made when separating non-thumb and thumb keystrokes. We first treat each detected thumb keystroke as a non-thumb keystroke, and estimate its touchpoint. We compute the distance of each "fake" touchpoint to the closet centroid of the non-thumb clusters (produced in the above step), and use this distance to cluster the thumb-based keystrokes. This produces roughly 10-15 clusters (depending on the attack video).

Identifying the 'delete' cluster(s). We declare a cluster as 'delete' (or 'backspace') if satisfying two conditions: (a) the cluster is at the very edge of the touchpoint map, and (b) the cluster instances were pressed multiple times consecutively. Upon detecting the 'delete' keystrokes, we follow its actual operation to remove their previous keystrokes.

2.1.5.4 Inferring Typed Content via HMM

Given a sequence of detected keystrokes and the clusters of those keystrokes, the attacker can apply a language-based Hidden Markov Model (HMM) [187] to estimate the typed content [278]. This is done by exploring the causal link between the keystrokes (and their hidden states representing the typed key) and the clusters. This inference requires computing a transitional matrix $\mathbf{T}$ and an emission matrix $\mathbf{E}$. $\mathbf{T}$ is a $N \times N$ matrix that defines the transition probabilities between the N hidden states, where N is the number of keys in the alphabet. Assuming the target types English, the attacker can pre-compute $\mathbf{T}$ using a large English corpus. For our attack implementation, we randomly select 40,000 sentences (52,000 unique words) from the CNN/DailyMail dataset [87], and set $N = 29$ to cover 26 letters, comma, period and space key. $\mathbf{E}$ is a $N \times M$ matrix, where M = # of clusters, $M < 50$ in our implementation. It measures the probability distribution of the N hidden states that produce the M clusters. HMM estimates $\mathbf{E}$ using a special Expectation-Maximization (EM) algorithm [30] to analyze the cluster data.

Given $\mathbf{T}$ and $\mathbf{E}$, the attacker applies the Viterbi algorithm [240] to infer the most likely hidden state sequence (or typed keys) for the keystroke sequence. Note that unlike [278], we do not pre-identify the thumb cluster as the 'space' key because we do not make assumption on the target's typing behavior. Furthermore, since EM runs local optimization [100], it can often converge to a local minima. Thus, we run several randomly initialized iterations of HMMs and select the one that produces the most high confidence keystroke labels (discuss in §2.1.7). We empirically confirm that this also leads to the lowest character error. Finally, since $\mathbf{E}$ is estimated from the keystroke data, we find that 150-200 words are generally sufficient under perfect clustering and keystroke detection.

2.1.5.5 Impact of Hand Tracking Noise

To examine the impact of hand tracking noise, we invited 2 volunteers (PA and PB) to type a randomly selected set of corporate emails (roughly 500 words, 28 sentences) on an iPad. This experiment is IRB-approved. We consider three hand tracking methods to extract fingertip data from the videos.

- *Perfect 3D tracking*: By tracking all 10 fingertips precisely in 3D, one should accurately detect when/where a fingertip touches the keyboard. We emulate this by assuming perfect keystroke detection and using the actual screen touchpoints recorded by the iPad as the input to the clustering algorithm.

- *Marker-assisted 2D tracking*: To emulate a high-performance 2D hand tracking, we place color markers near each participant's fingertips and locate each fingertip by its assigned marker on the video frame. The 2D tracking error is around 1cm. We also observe that the thumb tips are occluded in more than 32% of the video frames. Thus a practical attack should focus on non-thumb fingertips.

- *MediaPipe*: We use non-thumb fingertip data provided by MediaPipe. To avoid bias introduced by color markers, we ask our participants to type two sessions, one with markers and one not. We run MediaPipe on the video without.

As discussed earlier, the hand tracking error can affect keystroke detection, clustering, and HMM-based inference. Figure 2.5 plots, for user PB and the three tracking methods, the estimated touch-points of the detected non-thumb keystrokes, which are the input into the clustering algorithm. Here we color each point by its ground truth key input. For both marker-tracking and MediaPipe, the tracking error creates overlaps among different keys, and misdetects some thumb keystrokes as non-thumb ones (i.e., the points in the very bottom).

Next, Table 2.1 lists the detailed results, from keystroke detection accuracy, clustering accuracy to content accuracy. For a fair evaluation, we also list the content accuracy after applying a public spell check tool by Google Docs [92] (hereby referred to as GSpell for brevity). With perfect hand tracking, clustering is effective but not perfect because pressings near the key edge (compared to near the center) are harder to separate. The content recovery, evaluated as character error rate (CER) and word error rate (WER) (defined in §2.1.8.1), is also not perfect, mostly due to the error made by the HMM inference. But a standard spell check like GSpell can correct most of them. For the other two tracking methods, the tracking noise leads to 9-16% detection errors and 5-15% clustering errors, which propagate to the HMM inference component. Even after applying GSpell, the content accuracy is still low. The marker-assisted tracking is more accurate than MediaPipe, thus achieves better inference results. Together, these results demonstrate the severe impact of hand tracking noise and the significant difficulty facing a practical attack, which cannot assume perfect 3D tracking or marked-assisted tracking.

2.1.6 Self-supervised Inference on Video Data

After applying unsupervised inference on handpose data, the attacker obtains a noisy label on individual frames of the attack video. That is, for each detected keystroke, its corresonding video

Figure 2.5: The estimated touchpoints of the detected non-thumb keystrokes, using (left) perfect 3D hand tracking, (middle) 2D hand tracking using marker, and (right) MediaPipe. We mark each point by a color defined by its ground truth key entry.

		Detection		Cluster.	w/o GSpell		w/ GSpell	
		Miss (%)	Extra (%)	Acc. (%)	CER (%)	WER (%)	CER (%)	WER (%)
Perfect 3D	PA	0.0	0.0	99.8	3.1	16.2	1.6	7.0
	PB	0.0	0.0	99.6	4.2	22.5	1.8	8.9
Marker Assisted	PA	6.0	5.0	88.9	31.5	78.5	29.0	54.8
	PB	6.0	3.0	93.2	26.0	60.4	23.5	39.6
MediaPipe	PA	7.0	5.0	85.4	40.3	84.3	39.4	67.1
	PB	10.0	6.0	85.5	55.2	82.6	54.7	71.7

Table 2.1: Performance of unsupervised inference on handpose data, using three different hand tracking tools.

frame is labeled by its inferred key ('a'-'z', space, comma, period, as well as 'backspace'). The rest of the frames are not labeled (representing no keypress). While many video frames are wrongly labeled, our design seeks to identify the ones with high confidence labels and use them to train DNN inference models that operate on the entire set of video frames without applying hand tracking.

The key challenges facing this step include (1) how to identify high confidence labels, (2) how to train DNN models with limited training data to detect keystrokes and recognize their typed keys, and (3) how to suppress the impact of noisy labels during model training. We discuss them next in details.

2.1.7 Finding High Confidence Labels

Not knowing the keyboard layout, we filter keystrokes with high confidence labels using consistency check within each cluster and across clusters.

HMM label consistency within each cluster. HMM makes inference on the detected keystrokes by exploring how the keystroke clusters interact with the language model. Thus ideally, HMM should map keystroke instances in a cluster to a single key. But when the keystroke detection, touchpoint estimation and clustering are noisy, HMM often assigns different labels to keystroke instances in the same cluster to best match the language statistics. It can either incorrectly predict a legit and accurately clustered keystroke instance, or correctly predict a wrongly positioned and clustered keystroke instance. Considering this uncertainty, we propose to identify keystrokes with high confidence labels as those whose label matches the "majority label" of its cluster (i.e., the most popular label among the keystroke instances in the cluster).

Cross-cluster consistency check. Some detected keystrokes are false (i.e., no key is pressed) and some are identified with a far-off pressing finger. Together, these keystrokes can create multiple "spurious" clusters. HMM would wrongly predict most (if not all) instances in a spurious cluster, which the above intra-cluster check fails to address. Instead, we detect them using a *cross-cluster* consistency check, leveraging the fact that any valid clusters whose majority labels are the same should be right next to each other on the touchpoint map. Specifically, we first sort the clusters by size, and starting from the largest cluster, find its majority label, mark this label as "claimed" and move to the next cluster. If the cluster's majority label has already been claimed and its touchpoint area is not close to the cluster who has claimed the label, this cluster is marked as 'spurious' and no instance is selected.

When doing cross-cluster check, we make an exception for clusters whose majority label is the 'space' key. This is because most users use one or both thumbs to type the space key. Recall that in §2.1.5.3 we apply a separate clustering on the thumb keystrokes based on their non-thumb touchpoint (i.e., computing the touchpoint by treating it as a non-thumb keystroke), which creates

multiple clusters. If any of these clusters are labeled by HMM as 'space', it is most likely valid.

2.1.7.1 Training DNNs using Limited Data

After identifying keystrokes with high fidelity labels, we use them to train two DNN models, one to detect keystroke events and one to classify the key of the detected keystrokes.

Learning finger motion from a block of video frames. While our unsupervised inference operates on per-frame handpose data (to make the analysis tractable), both DNN models take as inputs a set of consecutive video frames (16 frames in our implementation). By operating on this short video segment, both models seek to discover and use rich finger motion features to make decisions, leveraging the labeled training data. Next, we discuss the detailed training process.

DNN-based keystroke detector. We implement the detection as a binary classifier. Since this detector will run against the entire attack video (a 10 min video has 360,000 frames), we choose a light-weight 3D-CNN model (ResNet-10 [86]) and apply transfer learning using a public teacher model, which is pre-trained using the EgoGesture dataset [274].

To train this binary classifier, we curate both positive and negative training data, leveraging the keystrokes detected by the unsupervised inference step without filtering. This is because the consistency check in §2.1.7 targets recognition consistency rather than keystroke detection consistency. For each detected keystroke, we find its corresponding video frame i and form a video segment of 16 frames, using 8 frames before i, i, and 7 frames after i. As such, the detected keystroke's frame is centered in the video segment. This video segment is labeled as 'positive.' To build 'negative' video segments, we apply a length-16 window on the video sequence between two consecutive keystroke frames. Finally, since the curated positive and negative labeled data will contain noise, we apply multiple techniques during the model training to identify/suppress noisy labels (discussed in §2.1.7.2).

At inference time, the trained binary classifier will scan through the entire video sequence, each time taking 16 consecutive frames as the input, and output a probability score. Thus near a

keystroke frame, multiple video segments will have high probability values for 'positive'. We use a peak detection method to identify the video segment where the keystroke frame is in the center.

DNN-based keystroke classifier. For this multi-class classifier, we use ResNeXt-101 [256], a well-known 3D-CNN architecture for video-based classification tasks. The training pipeline is similar to that of gesture recognition [112] – we apply transfer learning from a public teacher model (ResNeXt-101 pre-trained on the Jester dataset [145]). Our classifier takes as input a video sequence of 16 frames, and outputs a label out of the 29 classes ('a'-'z', space, comma, period).

We build its training dataset using the high confidence labeled data (identified by §2.1.7). For each keystroke with a high confidence label l, we build a video sequence of 16 frames (8 before and 7 frames after), and label this segment with l. To reduce training complexity, we crop each video frame uniformly to only include the area around the hands/keyboard/table (56×56 pixels). Like the above, the keystroke frame is in the center of the video segment.

2.1.7.2 Noise-aware Model Training

Since the training data for both DNN models can be noisy, we apply three kinds of noise-aware training techniques [20, 221] to suppress their impact on the trained models.

Preventing overfitting. We apply Mixup [270, 272] to mitigate overfitting in our 3D-CNN models, which applies data augmentation like interpolation to smooth the decision boundary between classes. Under our input configuration, we achieve this by linearly interpolating two random training inputs (i.e., blending each pair of video frames in two video segments) and their labels (in terms of their one hot vector representations).

Identifying trusted samples. DNN models are known to have memorization effect [221, 264, 102**?**], where noisy labels take longer to learn than clean labels and thus have higher loss during early training stages. We leverage this effect to emphasize learning on small-loss training samples (which are likely to have clean labels), by giving these samples with larger weights in the overall training loss computation.

Self-correcting noisy samples. The above technique can identify trusted samples and some untrusted (noisy) samples. Instead of discarding those noisy samples, we apply the concept of label refurbishing [193] to correct them using the knowledge that the current model has learnt. Specifically, noisy labels are refurbished as a linear combination of their actual model inference result and their noisy label. Here we adopt dynamic bootstrapping [**?**] to adapt the weight of the combination based on the training loss. As such, noisy labels receive more supervision from the model while the model itself learns more from cleaner samples.

In our experiments, we find that all three techniques are beneficial when training the keystroke detector, while the first technique is already sufficient to train a high-performance DNN classifier, possibly because we only use high confidence labels as its training data.

2.1.8 Experimental Evaluation

We evaluate our video based attacks using real-world user studies under a diverse set of conditions. All studies were approved by our Institutional Review Board (IRB21-1396). In this section, we organize our experiments and their results by four groups:

- **Attack performance under different scenarios**, including environments (indoor/outdoor, varying attack distances and blockages), keyboard devices (visible/invisible keyboards, varying size/layout, placed on desk vs. on lap), and content typed (§2.1.8.2);
- **Attack performance across 16 different users**, who have different typing behaviors and abilities (§2.1.8.3);
- **Contributions of individual components** (§2.1.8.4);
- **Attack complexity** in terms of computing time (§2.1.8.5).

2.1.8.1 Experiment Setup

Target (or victim) configuration. In each experiment, the default setting is that one study

participant sits in front of a table, where a keyboard device is placed on top of the table. The participants are free to adjust the chair and the keyboard device so that they can type at a comfortable position. By default, we ask them to type email sentences (about 500 words) randomly selected from the Enron corporate email dataset [52]. For a fair evaluation, we ask each participant to correct any typing errors using the backspace key, so that the final content matches the chosen content. As such, the actual keystroke data (recorded by the video) covers 26 letters, space, comma, period, backspace, and any other characters that they wrongly pressed and later corrected using backspace.

We do not apply any restriction to our participants except that the table and the keyboard device stay stationary during the typing session. Our participants are free to move and leave the seat during the study. In fact, to encourage natural movements, in our indoor experiments we placed a cart of snacks nearby, which they need to leave or move the wheeled chair to reach, and many did so.

Attacker configuration. We consider an attacker who uses their smartphone camera to record the keystroke video. The camera (after the initial calibration period) remains stationary during the typing session. We experiment with three iPhone models (iPhone X, 13Pro, 13Pro max) where the recorded video is consistently set to 720p at 60fps. The only exception is Scenario #4 in §2.1.8.2 where the video is set to 1080p at 60fps to enable outdoor long-range attacks. Both 720p and 1080p at 60fps are common camera settings for today's phones. Since our attack implementation uses MediaPipe to extract handpose data, the attacker needs to position the camera such that both hands are visible and that MediaPipe is working. This is done using the real-time hand tracking API provided by MediaPipe [147]. We find that the camera generally needs to be 10° above the keyboard. As such, the camera height will increase gracefully with the attack distance.

The attacker runs spell correction to further polish the recovered content. We build a fully automated spell correction using Google Doc's built-in function (referred to as GSpell earlier). While Google does not provide an API for this functionality, we develop a browser automation script using the Selenium library [206] to access the function. We query it at different levels of granularity (paragraph, sentence, phrase and word) to maximize correction effectiveness.

Evaluation metrics. We evaluate the effectiveness of the attack by comparing the typed and recovered content at the character, word and semantic levels, and the accuracy of recognizing individual keys (precision/recall). To provide context, we also include some samples of original and recovered text in Figure 2.8 in Appendix, where the CER value varies between 3.8% and 11.8%.

- **Character error rate (CER)**: We compute CER as the total Levenshtein Distance between the typed and recovered content divided by the number of characters in the typed content. The Levenshtein Distance between two strings [117] measures the minimum number of character-level operations (insertions, deletions and substitutions) required to convert one string into another.

- **Word error rate (WER)**: This is similar to CER except calculated at the word level. We use a public NLP tool [70] to compute WER, which applies dynamic string alignment to match words. We note that WER is a highly strict metric since one incorrect character is counted as a word error even when the word is comprehensible given the context.

- **Semantic content similarity (Similarity)**: We evaluate the semantic similarity between the typed and recovered content using CopyLeaks [54], a commercial tool for detecting plagiarised and paraphrased content. It reports a similarity score between 0-100% that accounts identical, minor changes and related meaning between any two documents. The similarity score is generally higher than 1-WER by capturing semantic correlation in words/sentences. However, we find that when WER is high (>50%), CopyLeaks produces a similarity score (much) smaller than (1-WER).

- **Per-key precision and recall**: Finally, we also compute the precision and recall of each character typed by the target to analyze the recovery rate for each individual key.

In §2.1.8.3, we also study the effectiveness of recovering websites typed during the study, in terms of the **top-k accuracy**.

2.1.8.2 Performance under Different Scenarios

We begin by a set of experiments to understand the feasibility of launching the proposed attack under different scenarios, exploring the impact of physical environment, attack distance, keyboard device/layout/movement, typed content, and attack observation window. For consistency, we invited a single participant for all the experiments.

Scenario #1: indoor lounge, varying attack distance. We consider typical indoor public spaces like a lounge or cafe, where the target sits by a table and uses a 12.9-inch iPad's on-screen keyboard to type corporate emails (from the Enron email dataset). We set four iPhone cameras at 0.8, 1.8, 2.4 and 3 meters away from the target. The camera heights are 0.3, 0.64, 0.64 and 1.09 meters above the target's keyboard. As mentioned earlier, the camera needs to be 10° higher than the keyboard for MediaPipe to function, thus the height increases with the attack distance. We use the camera's built-in optical zoom-in (1x for 0.8m, 2x for 1.8m and 3x for 2.4 and 3m) to capture the keystroke videos (60fps, 720p).

Table 2.2 summarizes the attack performance in terms of CER, WER and Similarity at the four attack distances, after the target has typed 28 sentences (501 words, 10.9 minutes). Across the content recovered from the four attack videos, the CER is consistently low (0.3-1.1%), while the WER varies between 1.8% and 6.0%. This variance is mostly caused by the difference among the four videos. But more importantly, the semantic similarity between the typed and recovered content is consistently high (>98%).

Scenario #2: indoor lounge, invisible keyboard. Next, we consider a more "extreme" case where the keyboard device itself is invisible to the attacker, e.g. the target uses a VR/AR system to view the keyboard in their own VR world and type directly on the table surface. We emulate this scenario using the well-known green screen method – covering the table with green cloth, changing the iPad's screen display to green hue, recording the attack video, and then keying out green colors. This allows us to remove the keyboard completely from the video while preserving the participant's hands. An example frame is shown in Figure 2.1 (d). Table 2.2 summarizes the

Distance (m)	Height (m)	CER (%)	WER (%)	Similarity (%)
Indoor, visible keyboard (iPad)				
0.8	0.3	1.1	6.0	98.8
1.8	0.6	1.1	5.2	98.0
2.4	0.6	0.3	1.8	99.4
3.0	1.1	0.4	2.0	99.4
Indoor, invisible keyboard, no visual cue				
1.8	0.6	0.5	2.6	99.6
2.4	0.6	1.1	3.8	98.0
3.0	1.1	1.0	4.0	99.2

Table 2.2: Attack performance in an indoor environment at different attack distances. The camera height (2nd column) refers to the relative distance above the keyboard.

# Human Passing per minute	CER (%)	WER (%)	Similarity (%)
0	1.1	6.0	98.8
5	5.8 ± 0.3	15.7 ± 1.6	92.9 ± 2.1
10	9.8 ± 0.5	22.1 ± 1.0	84.9 ± 1.5

Table 2.3: Attack performance when passing pedestrians block the attacker's view of the target from time to time.

attack performance. The mean CER, WER and similarity are $0.8 \pm 0.3\%$, $3.4 \pm 0.6\%$ and $98.9 \pm 0.7\%$ across the three distances. This result confirms that our attack *does not rely on any knowledge of the keyboard or any visual cue of the keystrokes on the keyboard.*

Scenario #3: indoor, blockage by passing pedestrians. In public spaces, passing pedestrians can block the attacker's view of the target from time to time. Using local measurements, we find that each blockage lasts roughly 0.2s or 12 consecutive video frames. Thus, we emulate on a given video the effect of P=5 and 10 passing pedestrians per minute, each blocking 12 consecutive video frames. The blockage instances are randomly distributed over time. The chosen P values represent one passing pedestrian every 12 and 6 seconds, respectively, which correspond to very busy environments. Table 2.3 summarizes the attack performance, in terms of mean and std for CER, WER, and Similarity, since we run 5 experiments per P value. We see that while blockage by pedestrians increases CER and WER, our attack can still recover most of the content at a high semantic similarity.

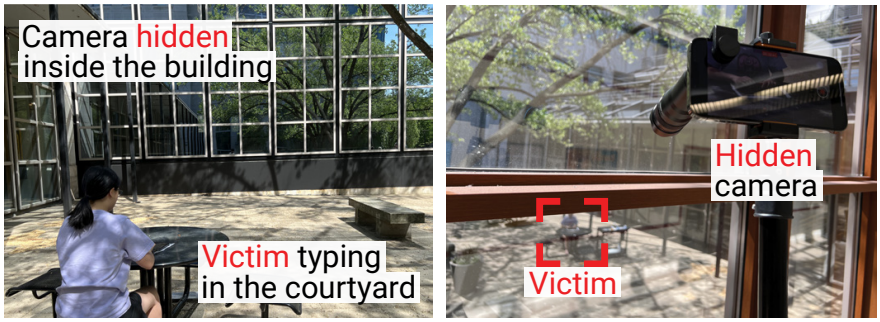

Figure 2.6: The experimental setup of our long-range, through-glass attack. The attacker videotapes the victim's hands, by placing a smartphone camera with a budget telephoto lens inside a nearby building's 2nd floor, behind the glass.

Scenario #4: outdoor, long-distance, through-glass attack. We also consider scenarios where the target is working in an outdoor courtyard, while the attacker records a video at a distance longer than the indoor scenarios. Specifically, we position a smartphone inside a nearby building's second floor, behind the glass, to record the target's typing. Here the target cannot observe the attacker (see Figure 2.6). The smartphone's camera is roughly 12 meters away from the target. We attach a budget telephoto lens (less than 60 USD) to the camera to help zoom-in onto the target's hands. Despite the complex lighting condition (sunlight, glass reflections and shadows), our long-range attack is still effective – recovering 82.4% of typed words and achieving a high semantic similarity of 87% (see Table 2.4). In parallel, we also set up another smartphone camera in the courtyard (4.5 meters away from the target), which is able to recover 96.8% of typed words accurately. Comparing the two videos (of the same typing session), we find that the 12m/through glass video appears more bland or dull, which likely affected the overall inference quality.

Attack condition	**Distance** (m)	**CER** (%)	**WER** (%)	**Similarity** (%)
Outdoor, open space	4.5	0.9	3.2	96.0
Outdoor, through-glass	12.0	5.2	17.6	87.2

Table 2.4: Attack performance in long-range outdoor scenario.

Scenario #5: varying keyboard type, size, layout. We are interested in understanding how our attack performs when the target uses different typing devices, keyboard layouts. We ask our partic-

ipant to type on three different portable keyboards: 12.9-inch iPad, 11-inch iPad, and a Bluetooth foldable keyboard purchased from Amazon. The first two are touchscreen keyboards and the third one is a more compact, rubberish keyboard. Results in Table 2.13 show that the attack is highly effective for all three keyboards.

We also explore the impact of keyboard's key layout on the attack. Here we consider the case where the target uses a secret layout that is significantly different from the default QWERTY layout. Specifically, we change[6] the 'a' key in the QWERTY layout to 'z', 'b' to 'a', 'c' to 'b', ..., 'z' to 'y'. The attack result located at the 4^{th} row in Table 2.13 shows that our attack is effective against this new, customized layout.

Scenario #6: on-lap keyboard. We are interested in the attack performance when the target types on an unstationary keyboard. Specifically, the target types for 15 minutes on a 12.9 inch iPad that was placed on their lap instead of a table. In the typing video, we notice small but observable keyboard movements. The attack result is shown in the last row of Table 2.13, which is comparable to the rest of the table. This shows that our attack can withstand minor keyboard movements.

Typing Device	**Dimension** (mm)	**CER** (%)	**WER** (%)	**Sim.** (%)
iPad Pro 12.9 inch	281 x 215	1.1	6.0	98.8
iPad Pro 11 inch	248 x 179	1.8	6.0	95.9
Foldable keyboard	210 x 85	0.9	4.2	98.8
iPad, secret layout	281 x 215	2.4	6.6	96.4
iPad, on lap	281 x 215	1.6	5.6	96.6

Table 2.5: Attack performance on different typing devices.

Scenario #7: varying content type and length. We examine attack performance when the target types different types of content. Beside corporate emails, we also consider machine learning paper abstracts (numerous technical terms), a Shakespearean play (Coriolanus) with numerous medieval period phrases, and medical patents (numerous medical terms). Table 2.14 summarizes,

6. Since it takes a lot of practice to type well on a new keyboard layout, we emulate typing on this new layout by the participant typing on the original QWERTY layout, but modifying the content to be typed to implement the layout change. For instance, to input word 'and' in this secret layout, the target just needs to press key 'b', 'm', 'e' in the original QWERTY keyboard.

for each experiment, the content type and length (# of words, # of sentences and video duration). Our attack remains highly effective across the four very different types of content. Furthermore, even by observing just 10 email sentences (199 words, 4.3 minutes of observation), our attack can successfully recover 87% of the typed words and achieve a high similarity score (94.5%).

	Content Length			Recovered Content		
	#Words	**#Sen.**	**Dur.** (min)	**CER** (%)	**WER** (%)	**Sim.** (%)
Paper Abstract	696	37	16.7	2.3	11.4	90.5
Shakespeare's Play	732	26	14.5	1.8	9.8	92.0
Medical Patent	708	36	18.2	0.7	6.5	97.3
Corporate Emails	654	40	13.9	1.1	5.5	99.1
	501	28	10.9	1.1	6.0	98.8
	199	10	4.3	2.8	13.1	94.5

Table 2.6: Attack performance when the target types content of different kinds and lengths. For corporate emails, the participant typed 40 sentences. We then shortened the video to match 28 and 10 typed sentences, respectively.

2.1.8.3 Performance across Different Users

Next, we examine our attack performance on different individuals, exploring the impact of typing styles and behaviors. We recruited 16 participants (P0-P15) locally (mean age=24.4 years, std=6.4 years; 6 females, 10 males). For consistency, we use the same 12.9in iPad as the typing device. The camera is placed roughly 0.8 meters away from the keyboard, 0.25-0.4 meters above the keyboard. The actual camera placement is chosen to obtain a stable result in MediaPipe (using its real-time API), which varies slightly across the 16 participants since they have different typing gestures. The typed content is a set of emails chosen from the Enron email dataset [52] ($\approx$ 500 words). Across the 16 participants, the (active) typing time is 10.7 $\pm$ 2.5 minutes. In addition to the emails, the participants typed 25 websites randomly extracted from the top-1000 websites in The Majestic Million[7].

7. The Majestic Million is a list of the top 1 million websites in the world: https://majestic.com/reports/majestic-million

		Email			Website	
	CPM	**CER** (%)	**WER** (%)	**Sim.** (%)	**Top-1** Acc.(%)	**Top-3** Acc.(%)
P0	169	0.7	3.4	99.6	100.0	100.0
P1	292	1.1	6.0	98.8	96.0	100.0
P2	284	2.3	8.4	97.2	96.0	100.0
P3	319	3.6	11.2	94.8	50.0	62.5
P4	291	5.0	12.1	93.5	100.0	100.0
P5	329	5.8	16.5	90.4	88.0	96.0
P6	313	5.2	14.9	87.6	92.0	100.0
P7	276	5.1	20.0	84.0	92.0	96.0
P8	342	8.0	25.5	83.4	96.0	100.0
P9	344	6.9	17.8	79.9	96.0	96.0
P10	331	11.6	32.9	71.0	92.0	100.0
P11	379	12.3	44.8	62.8	100.0	100.0
P12	308	13.6	35.5	59.0	68.0	76.0
P13	415	22.8	62.7	14.8	76.0	88.0
P14	198	1.0	3.6	97.4	96.0	100.0
P15	322	3.9	15.2	91.0	96.0	100.0

Table 2.7: Attack performance for all 16 participants (P0-15). The CPM column refers to their typing speed.

Observed typing behaviors. Our 16 typists display different typing behaviors. First, the number of fingers used varies – 13 participants use multiple fingers per hand while 3 use only two index fingers. The detailed finger usage is in Table 2.10 in Appendix. Second, the typing speed varies largely between 169 character-per-minute (CPM) and 415 CPM. Finally, 6 participants exhibit multi-touch behaviors, where they press the next key without releasing the current one, e.g. while the index finger is still pressing 't', the pinky presses 'a'.

Failed MediaPipe cases. We find that MediaPipe functions reasonably with some occasional flickers, except for 2 participants (P14 and P15). For both, MediaPipe failed to produce consistent results, where the detected fingers shift largely across frames. This is likely because these two users have very long, thin fingers. Instead of removing them from our evaluation, we apply the marker-assisted 2D tracking (see §2.1.5.5) by placing color tapes on their nails (except for the thumbs) and run our attack.

Overall attack performance. Table 2.15 summarized the attack results for all 16 participants. Our attack can effectively recover the typed corporate emails. The mean CER, WER and similarity

are 6.8% $\pm$ 5.8%, 20.7% $\pm$ 16.2% and 81.6% $\pm$ 21.7% between the inferred sentences and the ground truth. The attack performance does vary largely across the participants. For 10 out of 16 participants (P0-9), our attack achieves a low CER (0.7%-8%) and a high semantic similarity ($\geq$ 80% for 10 participants, $\geq$ 90% for 6 participants). For the two non-MediaPipe users (P14, P15), the accuracy is also high. We provide some samples of recovered and original text in Figure 2.8 in Appendix, at difference CER values (3.8% – 11.8%).

Our attack can also effectively recover websites typed during the attack window. Given the recovered text, we compute the edit distance to each website of the top-1000 websites to compute top-k accuracy. Across all 16 participants and websites tested, the top-1 accuracy is 89.6% $\pm$ 13.7%, and the top-3 accuracy is 94.7% $\pm$ 10.7%.

Three less effective cases: P11-13. Our attack is less effective on P11, P12 and P13. After a deeper study, we identified their unique behaviors that affect the attack performance.

- P13: *multi-touch, high speed typing* – Typing at a blasting speed of 415CPM, P13 used multi-touches constantly and the finger motion was much weaker than others. It is hard to detect and recognize these very subtle keystrokes.

- P12: *fake presses and 2-hand presses* – P12 exhibited frequent hesitation-retraction, i.e., press down towards a key, hesitate and then retract the finger(s) before hitting the key. The motion of these "fake" keypresses matches that of real keypresses. Also, P12 often presses keys using both hands simultaneously (i.e., multi-touch by 2 hands). Our current attack design does not consider this case.

- P11: *subtle thumb presses* – For P11, another high speed typist at 379CPM, our attack missed 'space' keystrokes more often than other users. This is because P11 typed 'space' with a thumb so subtle that there is very little motion at the non-thumb fingers. This contributed to the fast typing speed but also misled our attack.

Impact of character frequencies. We examine the inference accuracy on the character level. We

found that characters which are frequently used in the typed content are more accurately inferred. This is because they appear more often in the high confidence training data. Table 2.8 lists the character-level precision and recall, where we group characters in 6 buckets based on the number of appearances in the typed content ($\geq$500, $\geq$200, $\geq$100, $\geq$50, $\geq$25, $<$25). We report the mean $\pm$ std for both precision and recall across the characters in each bucket. While the frequency does affect the inference result, we only see visible degradation at the last bucket whose average frequency is 0.2% and only appeared 5 times in the content in average.

Characters	**Avg. # appear.**	**Avg. Freq.** (%)	**Precision** (%)	**Recall** (%)
Space	500	16.6	94 ± 5	93 ± 8
e, t	258	8.5	97 ± 3	95 ± 4
a, o, i, n, r, s	173	5.7	96 ± 4	95 ± 4
l, h, d, c, u, m, y, g	76	2.5	93 ± 5	91 ± 10
w, f, p, b, v	38	1.3	91 ± 12	90 ± 9
k, q, x, j, z	5	0.2	92 ± 14	61 ± 26

Table 2.8: Character-level precision and recall for all participants, bucketized by # of appearances of the character in the content.

2.1.8.4 Contributions of Different Components

To understand how each component contributes to the attack, we conduct an ablation study on P3 and P9, who display different performance levels. Table 2.9 lists P3 and P9's results. Both demonstrate the same trend. For a fair comparison, all the reported results were obtained after running the same automatic spell correction function.

These results show that the unsupervised inference on handpose data is highly sensitive to hand tracking noise. By selecting high confidence labels to train DNN detector and classifier, our attack reduces the CER from 22.5% to 3.6%, and boosts the semantic similarity from 9.1% to 94.8% for P3. We can also clearly see the contribution of individual components.

Impact of hand tracking tools. To examine the impact of hand tracking tools, we test our attack pipeline by replacing MediaPipe [269, 237] with IntagHand [119], a recently released hand

	Unsup. Infer	*DNN Detector*	*Label Filter*	*DNN Classifier*	*Noise Train*	**CER** (%)	**WER** (%)	**Sim.** (%)
P3	✓					22.5	59.4	9.1
	✓	✓				16.2	46.0	47.4
	✓	✓	✓	✓		5.0	16.1	88.7
	✓		✓	✓	✓	8.3	23.5	78.5
	✓	✓	✓	✓	✓	3.6	11.2	94.8
P9	✓					26.3	67.0	0.0
	✓	✓				23.7	61.2	18.8
	✓	✓	✓	✓		14.9	45.2	50.1
	✓		✓	✓	✓	10.8	34.6	73.0
	✓	✓	✓	✓	✓	6.9	17.8	79.9

Table 2.9: Contribution of each design component in the pipeline, tested on P3 and P9.

tracking tool. Given an attack video, we use the handpose data extracted by IntagHand as the input to our system, replacing those extracted by MediaPipe. The attack result is worse (13.6% CER, 43.3% WER and 60.4% similarity) than that using MediaPipe (0.8% CER, 3.4% WER and 98.6% similarity). A deeper look at IntagHand's tracking results shows that it produced much more frequent tracking errors than MediaPipe, likely because IntagHand is designed to target common in-the-air handposes like sign languages and conversational gestures. Such frequent tracking errors, despite having smaller amplitudes, are much harder to recover using our current design.

2.1.8.5 Attack Complexity

We test our attack pipeline on a server with an Intel Xeon Silver 4214 CPU and a NVIDIA TITAN RTX GPU. For a 12-min attack video (500 words), it takes 40 minutes to produce the final spell-corrected content. Specifically, the unsupervised inference takes 9.8 min (dominated by HMM's EM optimization), the DNN detector takes 10.3 min (8.3 min spent on model training), the DNN classifier takes 10.2 min (9.8 min spent on model training), and finally the automatic spell check (GSpell) takes 8.8 min. Here we exclude the MediaPipe extraction time since it can be done in real-time while recording the video.

2.1.9 Defenses

Our study demonstrates that a general, vision-based keystroke inference attack can succeed in realistic scenarios. Thus, users working in public settings should take precautions to protect their privacy from potential attackers. Beyond checking nearby areas for suspicious sensors and microphones, users should consider physical screens that block external line of sight to their hands while typing. This is likely the easiest and most effective defense against these attacks.

Another potential defense is to largely limit the amount of keystroke events, by augmenting keystroke typing with either predictive text[8] or voice-based input.

2.1.10 Limitations

We also note limitations and caveats to our work, which can be goals to be addressed in future work.

Dependency on hand tracking accuracy. Our attack design operates on today's hand tracking tools like MediaPipe, which cannot accurately track typing fingers at mm-level accuracy. Our design addresses this accuracy gap by a two-layer self-supervised learning pipeline. Future improvements in hand tracking (e.g. more accurate 3D joint estimation) might reduce the impact of this challenge and lead to simpler systems for vision-based keystroke inference. Similarly, our attack is less effective against high-speed typing targets, because the corresponding hand tracking results are less accurate.

Keyboard/camera movement. Our experiments assume traditional typing sessions where the keyboard is relatively stable, i.e., placed on a table or on the target's lap. Our results may not hold in active settings, e.g. a moving train or airplane experiencing turbulence. To mitigate the impact of movement, the attacker must precisely locate and track both the target's keyboard and the attack camera continuously.

8. Predictive text is a (mobile) input technology that suggests words a user may wish to insert. Instead of manually typing all the characters of a word on a keyboard, the user can choose from a list of suggested words.

Typing on smartphones and laptops. Our attack assumes access to a frontal view of the target's hands. When typing on a smartphone (i.e., holding the phone in mid-air and thumb-typing) or on a laptop with its screen up, the hands/fingers can be blocked by the device. Our current attack might be less effective under those scenarios. Future work can explore tracking fingertips from behind to estimate the keystroke locations. A more advanced hand tracking tool that withstands occlusion (e.g. palm blocking the fingertips) and object interaction (e.g. hands holding the smartphone) is needed for such attacks.

Infrequent keys. Our self-supervised approach requires data of valid keystrokes for a particular key to be recognized by the eventual DNN models. Keys that appear very infrequently in the video will have noisier curated training data, and less accurate recognition. On the other hand, their infrequency also means their errors will have lower impact on overall inference of typed text.

Hybrid inputs. Our attack assumed traditional typing scenarios which may not withstand hybrid input methods that augment keystroke typing with predictive text. However, the attacker can potentially counter this by obtaining a replica of the predictive feature and incorporating them into the HMM and DNN model training. We leave this to future work.

2.1.11 Conclusion

This paper describes our experiences developing a general, vision-based keystroke inference attack, which can infer content typed by a target using a single RGB camera. Our work differs significantly from prior work in that we do not rely on side-channel data or other assumptions beyond having a frontal view of the target's typing hands. While today's public hand tracking tools cannot accurately locate keystroking fingertips, our work proposes a novel 2-layer self-supervised learning pipeline to infer the typed content without requiring any prior data/knowledge of the target. Our user study results show such attacks can succeed in realistic scenarios, raising the immediate need for users working in public settings to protect their typing privacy, e.g. setting up a physical screen that blocks frontal views of their hands.

2.1.12 Appendix

	Left hand	Right hand	Multi Touch
P0	I	I	No
P1	T, I, M, R, P	I, M, R, P	No
P2	I	I	No
P3	I, M, R	I, M	Yes
P4	I	I	No
P5	I, M, R	T, I, M, R	No
P6	I, M, R, P	T, I, M, R, P	Yes
P7	I, M, R	T, I, M, R	No
P8	I, M, R, P	T, I, M, R, P	Yes
P9	I, M, R, P	T, I, M, R	No
P10	I, M, R	I, M, R	Yes
P11	T, I, M, R, P	I, M, R, P	Yes
P12	I, M, R	I, M, R	No
P13	I, M, R, P	T, I, M, R, P	Yes
P14	I, M	I, M, R	No
P15	I, M, R P	T, I, M, R	No

Table 2.10: Typing behaviors of our 16 participants (P0-P15). We refer to individual fingers as Thumb (T), Index (I), Middle (M), Ring (R) and Pinky (P).

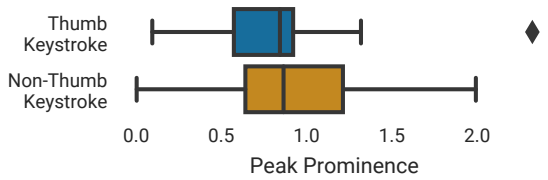

Figure 2.7: Distribution of negative acceleration's peak prominence for thumb-based and non-thumb keystrokes.

Inferred Text	Ground Truth	CER (%)
once we know exactly what contracts we are looking at we can fine tune the calculation	once we know exactly what contracts we are looking at we can fine tune the calculation	0
each trader will be us ed to manage their individual position and profitability goals for the simulation n	each trader will be asked to manage their individual position and profitability goals for the simulation	3.8
traders will be managing their individual booms and associates product	traders will be managing their individual books and associated products	5.5
the acatt ached draft is fairly legalistic in tone	the attached draft is fairly legalistic in tone	6.3
the lle ongoing uncertainty about our future coupled with the contestant media scrutiny makes the situation difficult for all of uw	the ongoing uncertainty about our future coupled with the constant media scrutiny makes this situation difficult for all of us	7.1
hbur message will be scanned an nd hec ed for viruses prior to requested release	your message will be scanned and checked for viruses prior to requested release	7.6
i attach a letter of intent ent w ich i hope covers all the points we discussed this m ing	i attach a letter of intent which i hope covers all the points we discussed this morning	9.9
os one of the enhanced sec rit measur s we have ecentl empo edm we will be e ch c ing emp o ee badges at the entrance to the ballr o om	as one of the enhanced security measures we have recently employed we will b e checking employee badges at the entrance to the ballroom	11.8

Figure 2.8: Examples of final recovered text compared to ground truth.

2.2 A General Keystroke Inference Attack in Virtual Reality

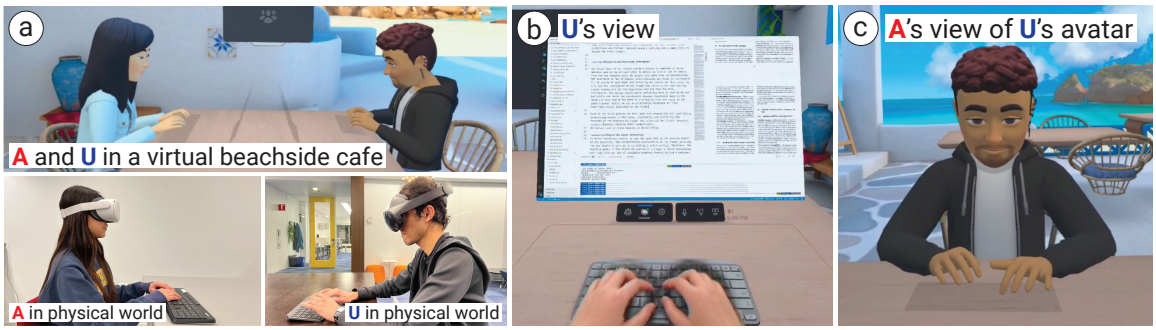

Figure 2.9: An illustrative example of keystroke inference attacks in the shared VR space. (a) In this scenario, U is replying to emails while enjoying the immersive experience of a virtual beachside cafe. U types comfortably and swiftly on a physical keyboard that is wirelessly connected to the VR headset. This is a typical setup of a VR office. Adversary A is another VR user in the same virtual beachside cafe. (b) From their VR headset, U sees the typed content on a virtual screen and a rendered view of their own hands. (c) A's view of U's avatar, displayed by A's VR headset. Using this information, A seeks to recover the content that U is typing. Credit: The VR scenes are screenshots taken when running the Horizon Workroom application from Meta [150].

2.2.1 Introduction

Virtual Reality (VR) offers a whole new dimension of online interaction significantly more immersive and lifelike than those offered by existing services. By eliminating the need for physical proximity and formal attire, VR cultivates a more convenient, comfortable, and efficient environment for work, play, and social interaction. Thus, VR is increasingly adopted for both personal and professional purposes [149, 186]. In particular, remote work through VR has witnessed remarkable growth. Numerous companies actively promote its adoption with their remote work platforms, including Meta's Horizon Workrooms [150] and VR applications and startups such as Valve [2] and vSpatial [246]. Apple's recent launch of Vision Pro [17] serves as further validation of the substantial interest and investment being made in VR-based remote work.

Because users interact with others using their digital representations or avatars, VR also provides them with a unique sense of privacy and added security. Representation through their digital selves can be particularly attractive for sensitive professional/work activities [80]. In general,

VR users can customize how they present themselves and choose what they want to show of their physical environments. This "natural filter" can help prevent the unwanted leakage of private information without the user's knowledge. For example, it might block traditional side-channel attacks, where an attacker can use physical observations of a user or their devices to steal information, e.g., screen peeking, shoulder surfing, and keystroke inference [278, 265, 261].

In this paper, we challenge this assumption by investigating the feasibility of keystroke inference attacks in shared VR environments, where one VR user may attempt to recover the content typed by another user by recording and analyzing their avatar's actions. Since typing is the primary input method for sensitive information to computing systems [260], successful keystroke inference attacks can cause significant damage, such as unauthorized disclosure of confidential or classified data that may lead to breaches of personal privacy or even financial losses. Compared to physical attacks, keystroke inference attacks over VR can be much more powerful, because a single attacker can target numerous remote victims without physical proximity constraints.

Ironically, we show that VR's most appealing properties, its immersive and interactive nature, are the source of VR users' vulnerability to keystroke inference attacks. Specifically, immersive experiences in VR rely on each individual's *expressive* avatars [50, 93], which display the user's body movements, hand gestures, and facial expressions. When an individual performs physical actions such as grasping objects, clapping hands, or typing on a keyboard, their avatar demonstrates an approximate, digital version of those hand movements, which are visible to other users in the same virtual space. Thus, VR actually "facilitates" side-channel attacks by eliminating the need and costs of physical sensing.

Our work shows that, while the accuracy, resolution, and granularity of these VR movements are far from those of physical observations (e.g., a physical video recording of the hands), intelligent attackers can still use them to extract detailed information "carried" by the movements, e.g., keys being typed. Therefore, the same feature of VR that drives growing usage and adoption is also enabling a real-world threat against user privacy and information security.

In this paper, we describe the design, implementation, and evaluation of keystroke inference attacks in shared VR spaces. Figure 2.9 plots an illustrative scenario of the attack. This attack occurs entirely in the virtual space between avatars, without any sensing or data from the physical world. An attacker in the same virtual space as the target gathers the noisy digital representation of hand movements displayed by the target's avatar, and uses it to reconstruct the typed data. We hereby refer to these digital movements as **telemetry** of the VR user.

An effective attack faces two unique challenges: (i) the excessive amount of noise and inconsistency in the telemetry data, and (ii) the lack of knowledge and labeled data on the target user and their physical devices, as they are physically isolated from the adversary. These factors, combined with the inherent complexity and variability of human typing, make it challenging to extract meaningful information from the available data. Commonly used transferability-based attacks (i.e., training a model on known users and applying it to others) are ineffective because an individual's typing behaviors are unique [28, 37, 74] and do not transfer to others [261].

We address these challenges by developing a componentized, self-supervised learning pipeline. We decompose the difficult task of keystroke inference into three sequential components, responsible for detecting keystroking events, identifying finger used, and recognizing the key, respectively. For each component, we first run statistical analysis on simplified telemetry data to produce initial inference results, leveraging general understandings of human typing. Then in the second component, we use these results to curate labeled training data and train a DNN classifier (i.e., a transformer), one that learns the intricate relationship between keystrokes and the full telemetry data. The output of this DNN classifier is then fed to the final component. Using this componentized design, our attack can progressively extract, filter, and learn important keystroke information from the noisy telemetry data, while minimizing the spread of error in the pipeline.

Our attack also supports flexible placement of the adversary in the virtual space, whether it is sitting at the same table (Figure 2.9), across the room, or floating in the air on a balloon. This is achieved by applying a coordinate transformation to normalize telemetry data seen by *arbitrary*

VR users onto a unified coordinate system so that a wide variety of attacks can be executed using a single attack pipeline.

Our work makes four key contributions.

- We identify three forms of keystroke inference attacks that can be executed by a VR adversary in the same virtual space as the target user. These attacks consider different levels of information access available to the adversary, including the original 3D telemetry collected by the target's headset, the transformed 3D telemetry sent to the adversary to render the target's avatar, and its 2D version extracted directly from the adversary's screenshots of the avatar (i.e., Figure 2.9(c)).

- We design an effective attack pipeline to infer keystroke content from noisy attack data, using a componentized self-supervised learning approach.

- We perform IRB-approved user studies to evaluate the proposed attacks under a variety of VR scenarios and settings, varying avatar distance and placement, keyboard device, and typing content. Moreover, we test the attacks on 15 users. On average, our attacks accurately recognize 91.3% of the typed keys, and the recovered content retains 71.5% of the meaning of the typed content (see Table 2.15).

- We explore potential defenses and find that adding zero-mean Gaussian noise to the telemetry data can reduce attack effectiveness while preserving the immersive experience to some degree.

To the best of our knowledge, our work is the first to explore and demonstrate the feasibility of keystroke inference attacks in shared VR environments. Our work identifies the crucial challenge of designing VR systems that can provide immersive and engaging experiences while ensuring personal and information security. Also, our attack methodology represents a substantial departure from the conventional understanding of transformers. We hope our results can inspire more self-supervised, transformer-based attack/defense designs.

2.2.2 Background and Related Work

In this section, we first describe existing keystroke inference attacks in both physical and VR environments. We then discuss the key features adopted by VR to create immersive experiences (which are the base of our proposed attack) and the general landscape of privacy threats in VR.

2.2.2.1 Existing Keystroke Inference Attacks

In a keystroke inference attack, an attacker can reconstruct a target's typed content without control over their input devices. We divide existing attacks into three categories: (i) physical attacks that place sensors near the target to capture their typing actions, (ii) remote attacks analyzing the target's network traffic, and (iii) attacks where the target is a VR user.

Physical attacks. By physically placing sensors (e.g., cameras, microphones, inertial measurement units (IMU), electromagnetic (EM), and radio frequency (RF) devices) near a target, attackers can record data related to the target's typing actions. The recorded sensor data includes audio [278] and video [27, 122, 210, 265, 189, 258], as well as vibration, EM [104], WiFi and LTE measurements [125, 120, 267]. As pressing different keys causes changes in signals recorded by sensors, attackers can use statistical or machine learning analysis to determine keystroke content. For example, a well-known audio-based attack [278] placed a microphone next to the target to capture key-specific sounds generated by typing on a mechanical keyboard, and trained a machine learning model to identify key entries from the recording. Follow-up works replace microphones with other sensors like IMUs, WiFi/LTE radios, or EM sniffers. Other examples use cameras to capture the screen and the pressing fingertip reflected by the target's eyes or eyeglasses [189, 258], or to record the reflections around the pressing fingertip produced by a reflective typing surface, such as a tablet [265].

Physical attacks are mostly ineffective in the VR attack settings where the adversary, a VR user, cannot place or access sensors or side channels (e.g., eye reflection or surface reflection) in the target's physical space. Moreover, many physical attacks require knowledge of the keyboard

and its layout, which are inaccessible to the VR attacker.

Remote attacks by traffic analysis. Song et al. [220] show that by analyzing encrypted SSH traffic, attackers can detect when a user is typing a password, and estimate the password length and content. This is possible because SSH pads data for encryption using an 8-byte boundary and sends packets upon each keystroke in interactive mode. Thus, attackers can infer password length by counting packets and estimate its content using packet inter-arrival times. On the other hand, this attack is only applicable to SSH sessions.

Attacks against VR users. Recent works have examined keystroke inference attacks against VR users [126, 19, 136, 234]. They all target VR users who use a virtual keyboard to input text. To enter a key, the user may either rotate their head to move the cursor on the virtual keyboard and make a specific hand gesture to select the key [126], or type by moving two fingers (one per hand) in the air [234] or by pointing handheld controllers [19]. These virtual keystrokes result in simple and slow typing movements that can be monitored by attackers using methods such as deploying a stereo camera [126] or a pair of WiFi devices [19] near the target, or by installing malware directly on the target's VR headset [126, 234]. These attacks also assume perfect knowledge of the "virtual keyboard" layout and screen location, and often labeled data from the target.

2.2.2.2 Creating Immersive Experiences in VR

Expressive avatars. The most compelling aspect of VR is its capacity to generate immersive experiences through expressive avatars. Expressive avatars foster a heightened sense of presence and connection by closely emulating users' real-life movements and expressions. To build such avatars, modern VR systems (e.g., Meta Quest 2/3/Pro, HTC VIVE XR Elite) deploy real-time hand tracking and isomorphic rendering, so users can see and manipulate virtual objects with their own hands. The rendering of hand movements and gestures in their avatars adds an additional layer of immersion, allowing users to communicate, gesture, and interact with others naturally in the VR environment. This level of realism and expressiveness provides a true sense of presence

and social interaction [242].

Hand tracking. Hand tracking in VR tracks the position and orientation of a user's physical hands. Today, it is achieved using multiple cameras and IMUs deployed on the user's VR headset [244]. The accuracy of hand tracking is influenced by various factors such as the relative position of the hands and headset, the number and placement of IMUs and cameras, lighting conditions in the environment, and the type and speed of hand movements. Even with advanced VR systems, some level of error, latency, or jitter in tracking may still occur, particularly when the user's hands are moving rapidly.

Currently, the accuracy of hand tracking falls short of the level required for precise keystroke recognition [67]. This is due to the fast and intricate finger movements that are specific to each user and constantly changing, along with the natural obstruction of the headset-mounted cameras by the user's hands. To overcome this challenge, commercial VR systems utilize a physical keyboard as the input device, while researchers suggest incorporating additional sensors, such as IMUs on the user's wrists [148], to improve the capture of finger-tapping movements.

2.2.2.3 Privacy Threats and Defenses in VR

As VR systems can gather large amounts of data, including user activity, location, and behavior, they are inherently susceptible to privacy attacks that steal or misuse personal information or sensitive data. A recent study [**?**] summarizes the landscape of VR privacy threats and defenses by reviewing 68 publications, which lists 30 VR attacks, including the two keystroke inference attacks [126, 19] discussed in §2.2.2.1. The other 28 attacks focus on revealing the user's demographics (e.g., age, gender, ethnicity) or sensitive attributes (e.g., emotion, physical and mental health, and wealth).

2.2.3 Keystroke Inference Attacks in a Shared Virtual Environment

We study a new, VR-based scenario for keystroke inference attacks where both the target and the adversary are VR users in the same virtual environment (see Figure 2.9). While the target types, the adversary seeks to recover the typed content using information displayed by the target's avatar. Without the constraints of physical proximity to the target, a single adversary can attack many remote victims, increasing the potential reach and impact of the attack. Next, we discuss the motivation behind the attack and the threat model.

2.2.3.1 Motivation and Real-World Implications

Our attack is driven by two important trends in VR. *First*, productive typing on a full physical keyboard is becoming a common mode of input in VR systems. Compared to virtual keyboards, physical keyboards offer precision, comfort, familiar use, and haptic feedback [33]. Popular VR platforms like Oculus Quest 2/3/Pro and HTC VIVE and VR apps like vSpatial and Immersed already support physical keyboards through Bluetooth connection [1, 2, 246, 97]. Many VR users also prefer to use them to input text, especially when working for extended period of time [33]. As users can now type efficiently and professionally in their VR workspace, text input in VR will quickly grow to encompass emails, documents, and other text-based materials that may contain sensitive information and personal data.

Second, VR enables immersive interaction among people in shared virtual environments where each VR user is represented by their own avatar rather than their physical self. This contrasts with the well-known privacy issue in physical spaces where people are directly visible to each other.

These two developments prompt the following question, which motivated our study: ***"Is it safe to type sensitive information in a shared virtual environment, since VR users can only see each other's avatars?"***

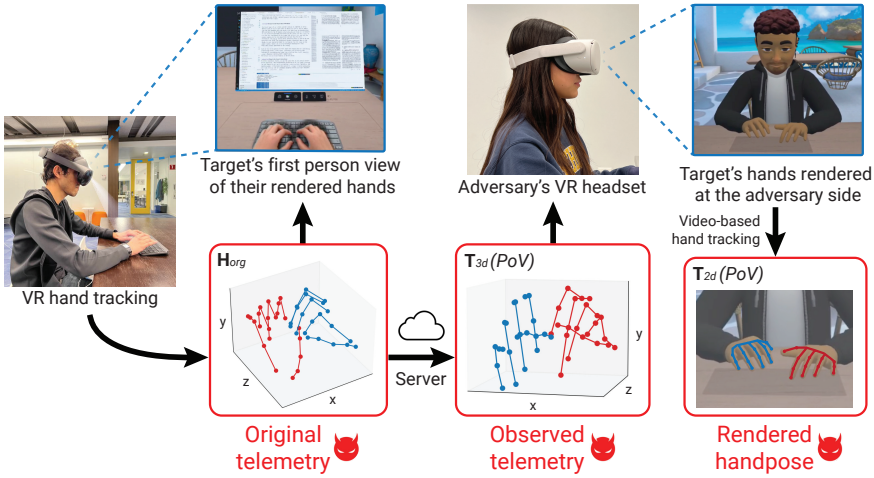

Figure 2.10: The three keystroke inference attacks considered by our work, operating on the original telemetry, the observed telemetry (used to render the target's avatar), or the video of the target's avatar displayed on the adversary's VR screen.

2.2.3.2 Threat Model

We consider a VR user $\mathcal{U}$, who is doing work in VR by typing on a physical keyboard wirelessly connected to their VR headset. We consider an adversary $\mathcal{A}$ who is another VR user sharing the virtual environment with $\mathcal{U}$. To enable seamless interaction, the VR platform uses an expressive *avatar* to represent each user in the virtual environment, which displays their nonverbal behaviors such as body movements and hand gestures [93, 50].

We assume that $\mathcal{A}$ can *only gather data related to* $\mathcal{U}$*'s avatar*. To render $\mathcal{U}$'s avatar to be seen by $\mathcal{A}$, the headset of $\mathcal{A}$ will receive a continuous stream of $\mathcal{U}$'s telemetry data, i.e., $\mathcal{U}$'s handposes. We note that although hand tracking is a built-in feature of modern VR headsets, the regulations governing the access and sharing of its data are still in development. Upon user consent, VR apps can access, store, and stream this information to render avatars on other devices. Therefore, we investigate various attack scenarios by varying the adversary's level of information access and the design of the VR app. Following this process, we identify three versions of the attack, defined by the telemetry data gathered by the adversary (see Figure 2.10).

- **Attack I: Original telemetry attack operating on $\mathbf{H}_{org}$.** This represents the strongest

adversary, one who has access to either the target's headset or the VR rendering server [**?**]. Here $\mathbf{H}_{org}$ is the 3D telemetry data on $\mathcal{U}$'s hands collected by $\mathcal{U}$'s headset, which is extracted from a top-down (or bird's-eye) view of the hands.

- **Attack II: Observed telemetry attack operating on $\mathbf{T}_{3d}(PoV)$.** Here the adversary obtains $\mathcal{U}$'s avatar data by setting up a virtual camera. To render the virtual hands of $\mathcal{U}$'s avatar to be seen by $\mathcal{A}$, the VR system needs to transform the original telemetry data $\mathbf{H}_{org}$ to screen coordinates of $\mathcal{A}$'s virtual camera. The latter is defined by PoV, $\mathcal{A}$'s camera point-of-view (PoV) with respect to $\mathcal{U}$. For example, when $\mathcal{A}$'s avatar (thus the camera) directly faces $\mathcal{U}$, i.e., $PoV = (0, 0)$, the observed telemetry $\mathbf{T}_{3d}(0, 0)$ captures a frontal-view of $\mathcal{U}$'s hands. Depending on the VR system/app design, $\mathbf{T}_{3d}(PoV)$ can be calculated by the VR server and sent to $\mathcal{A}$'s headset, or by $\mathcal{A}$'s headset after receiving $\mathbf{H}_{org}$. Here we note that the latter makes it possible for $\mathcal{A}$ to gather $\mathbf{H}_{org}$ directly from $\mathcal{A}$'s VR device and launch the Attack I mentioned above.

- **Attack III: Rendered handpose attack using $\mathbf{T}_{2d}(PoV)$.** The attack operates on a 2D projection of $\mathbf{T}_{3d}(PoV)$ used by Attack II such that the depth-to-camera data is removed. This represents a limited adversary who has no access to any telemetry data, but instead records $\mathcal{U}$'s avatar shown on $\mathcal{A}$'s VR screen and applies a hand tracking tool to extract the 2D handpose per video frame.

In our attack design, the only assumption is that the attacker is aware of the language used by the target (English in this study). More importantly, the attacker has **no** other information about the target. This means the attacker lacks knowledge of the target's keyboard layout and position, does not have access to labeled data or prior observations of the target, and does not have any additional side-channel information besides the avatar telemetry/handpose data previously discussed.

Extension to virtual keyboards. Note that our proposed attack naturally extends to future VR systems allowing controller-free virtual typing on a surface. The attack works because it requires

only handpose data and no knowledge of a "keyboard." While such virtual typing is infeasible today due to insufficient accuracy in hand tracking, it may become a reality in the near future as VR hardware/technology advances. Ironically, improvements in the accuracy of VR hand tracking systems will only increase the potency of our attack.

2.2.4 Design Challenges and Baseline Solutions

The proposed attack scenario is unique because the adversary only utilizes telemetry/handpose data of the target's avatar. However, this also presents significant difficulties in constructing a successful attack due to the high levels of noise present in the data. In this section, we address this challenge through a thorough analysis of the attack data and an examination of potential attack designs. Our discussion focuses on the original telemetry attack ($\mathbf{H}_{org}$) because $\mathbf{T}_{3d}$ and $\mathbf{T}_{2d}$ are lossy transformations of $\mathbf{H}_{org}$.

2.2.4.1 A Closer Look at the Telemetry Data

We examine the 3D telemetry data $\mathbf{H}_{org}$ produced by the Oculus Quest Pro headset, using bird's-eye views of the wearer's hands captured by the headset's four cameras. At a sampling time t, the 3D telemetry is represented by (x, y, z) for each of the 46 joints on both hands (23 joints per hand). This 3D coordinate system is arranged by p, the physical position of the headset measured at the start of the VR session, and remains unchanged throughout the session even as the user's head moves around. Therefore, given the (x, y, z) coordinate of a joint, y is the depth of the joint to p, while the (x, z) are the deviation from p in the horizontal plane. Figure 2.12 (a) shows a sample instance of $\mathbf{H}_{org}$, where the left pinky is pressing a key. Since the physical keyboard is placed on a horizontal plane, the y value captures the relative *height* to the keyboard while the xz values indicate the *positions within* the keyboard.

Virtual vs. physical hands. Figure 2.11 plots a sample observation seen by $\mathcal{U}$, where the *virtual* hands (rendered by $\mathcal{U}$'s headset from $\mathbf{H}_{org}$) are overlaid on top of the physical hands. The virtual

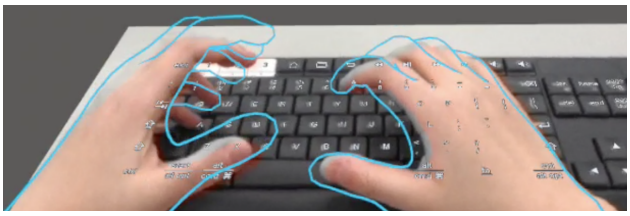

Figure 2.11: The virtual hands rendered by VR headset, overlaid on top of their corresponding physical hands.

Figure 2.12: (a) A sample instance of $\mathbf{H}_{org}$ where the left pinky is pressing a key. (b) VR collected (x, z) touchpoints from 2 participants. The corresponding ground truth key positions are marked with the same color. Top-15 frequent letter samples are shown. (c) Depth of sequential keystrokes when 'e' is typed by Participants 1 and 2. Normalized depth error puts the ground truth at zero and scales by the height of an individual key. Standard Deviation for Participant 1: 1.639. Standard Deviation for Participant 2: 1.853.

hands do not align with the physical hands, especially at the fingertips, demonstrating that the telemetry data is noisy. This is expected as hands can often obstruct fingertips from the view of the headset cameras and typing on a physical keyboard is rapid and delicate.

Error analysis. We present an in-depth analysis of the noise in the telemetry data. As it is challenging to obtain ground truth positions of hand joints, we choose to analyze the fingertip coordinates when the user is pressing a key. Accurate recording of timing and keystrokes is ensured by the physical keyboard. For each keystroking event, we determine the corresponding pressing fingertip by manually reviewing an external video of the typing hands.

Noise in (x, z)**:** Figure 2.12 (b) shows, for two users, the spread of (x, z) values of pressing fingertips (reported by $\mathbf{H}_{org}$) on top of the actual physical keyboard. Each press point is marked by the color of the key being pressed. For clarity, we only show the top 15 frequently used letters. Both users exhibit clear deviations, with variations among keys and users. The physical size and

spacing of keys do result in some level of separation in the xz-plane of pressing points.

Noise in y**:** The noise effect on the depth y value is greater due to the tendency of users to rest their hands on the physical keyboard and only slightly lift their fingers to press keys. This results in a small depth difference between pressing and non-pressing fingertips. Figure 2.12 (c) plots, for two users, the error of depth measurements for pressing the letter 'e', normalized by the key height (4.4mm). These results come from 300 instances of typing 'e', extracted from a random typing session. For each instance, we find the pressing fingertip's y value, subtract from it the ground truth depth of the keyboard, and divide the residual by the key height. Ideally, the values should be very close to 0 (i.e., the blue line marking the ground truth depth of the keyboard). The actual results demonstrate significant deviations and display a user-specific bias. For both users, over 50% of the typing instances have depth errors higher than $2\times$key height.

The same depth noise affects non-pressing fingers, making it challenging to identify both keystroking events and their corresponding pressing fingertips. To study the impact on identifying the pressing fingertip, we compute, for each keystroking event, the deviation from the "lowest" non-pressing fingertip's y to the pressing fingertip's y. Ideally, the deviation should be positive and $\geq 1\times$ key height, since the pressing fingertip is the lowest across all fingers. Yet we find that, for both users, **more than 30%** of the instances have negative values, i.e., at least one non-pressing fingertip is wrongly reported as pressing lower than the pressing fingertip.

Summary. These findings collectively demonstrate the presence and impact of excessive noise in the 3D telemetry data $\mathbf{H}_{org}$, particularly for the y values. We also note that, for the other two forms of handpose data ($\mathbf{T}_{3d}$ and $\mathbf{T}_{2d}$), the errors are exacerbated by the lossy transformations from $\mathbf{H}_{org}$.

2.2.4.2 Exploring Attack Design Options

In our quest to find an effective attack, we attempted the following methods, but none proved to be successful. We present them below and summarize key limitations of each.

Denoising telemetry data. An important question is whether techniques such as data pre-processing and denoising can enhance the quality of the telemetry data. Unfortunately, this is challenging as the noise in the telemetry data follows a Gaussian distribution. For instance, noise in the depth y value of the pressing fingertip is confirmed to be Gaussian through KS goodness-of-fit tests, with a p-value greater than 0.5.

Manual inspection. A curious attacker can manually inspect each telemetry frame to identify keystroke events and the corresponding pressed key. However, we attempted this and found that the random noise in the telemetry data makes it challenging for the human eye to accurately identify keystrokes and the finger responsible.

Attack by transferability. Another option is to leverage transferability across users [234], i.e., collecting accurately labeled keystroke/telemetry data from a set of users, training a DNN inference model, and applying it to infer keystroke content of other target users. We examined this attack among 7 users, applying leave-one-out cross validation. In each round, we selected 6 users, trained a supervised DNN keystroke classification model using labeled telemetry data from them, and tested the trained model on the one new user. We evaluated the attack by comparing the typed and recovered content (after apply spell correction). The results were consistent: the attack on a new user unseen during training produced a large character error rate (56% $\pm$11%) and could not recover any meaningful content. But on trained users, the models can correctly detect and recognize more than 90% of keystrokes (i.e., $<$10% character error rate). This aligns with other keystroke studies, which show that human typing behaviors are user-specific and transferability-based attacks are ineffective [234, 261].

Statistical analysis of fingertip motion. With no labeled data from the target to train an ML model, one can apply unsupervised inference by studying the target's fingertip movements. Specifically, one can extract fingertip positions from the telemetry data, and apply statistical analysis to detect the keystroking events and identify the corresponding pressing fingertip. However, given the abundant noise in the telemetry data, the attack is ineffective (see our experiments in §2.2.7.5).

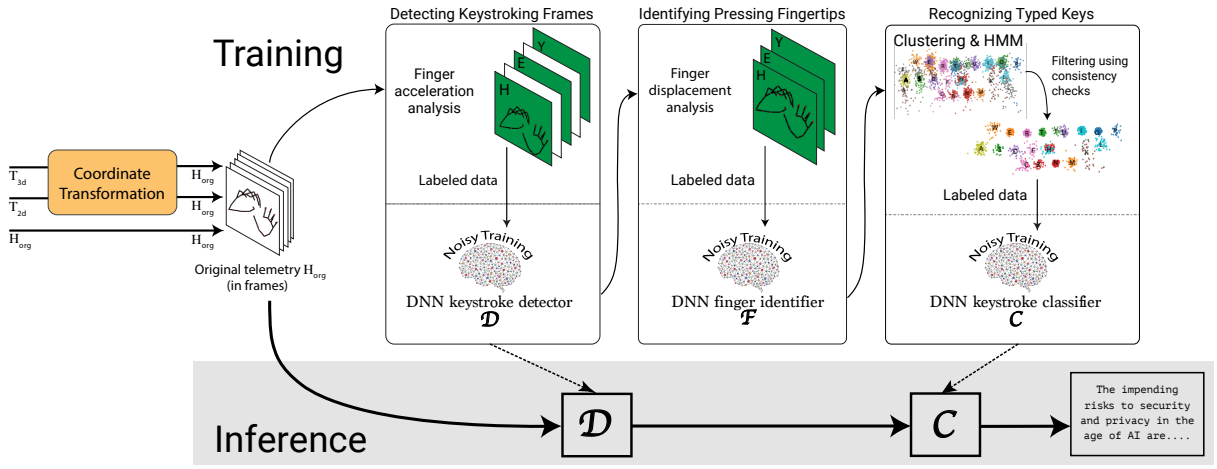

Figure 2.13: The end-to-end attack design, using a componentized self-supervised learning pipeline

Our finding aligns with existing studies on VR keystroke recognition [148]: the depth y of the VR hand tracking lacks the required accuracy to identify keystroking events and the pressing finger. To compensate for these errors, researchers propose placing extra IMUs on the user's wrists and collecting labeled data to train a DNN model for finger identification [148]. Yet both are infeasible under our attack scenario.

2.2.5 Our Proposed Attack

In this section, we present the proposed attack design. We begin with the key insights behind our design and a summary of the attack pipeline. Later in §2.2.6 we delve into the individual components and the implementation of the attack.

2.2.5.1 Design Insights

Our exploration of the transferability-based attacks (in §2.2.4.2) revealed that, with accurately labeled training data from the target user, a sophisticated DNN model can be trained to learn the intricate connections between keystrokes and noisy 3D telemetry data, allowing it to correctly recognize over 90% of keystrokes. This suggests that an effective attack is possible if we can self-label some of the target's 3D telemetry data and use them for model training. Such labeling is done

on the target's data since transferability-based attacks are ineffective.

The concept of self-curating labeled training data from the target data is an established technique in self-supervised learning [12, 277]. A recent work [261] applies this concept to recognize keystrokes from a video recording of the target's physical hands. This is done by first applying hand tracking to locate the target's fingertips in each video frame, followed by a HMM-based statistical analysis to estimate labels for a small subset of the video frames. These labeled data are then used to train CNN models to recognize the keystrokes captured by all video frames.

But when applying the method of [261] to our attack scenario, we observe significant failures (see Table 2.15). This is because the amount of noise included in our attack data is higher than that of [261], i.e., our attack data is the noisy hand tracking results estimated by the VR headset, while [261] uses the raw video capturing the target's physical fingers as the input to their CNNs. This qualitative change renders the method of [261] ineffective and requires a new approach.

Self-supervised learning of noisy handposes using task-specific transformers. To handle the extremely noisy telemetry data, we design our attack based on two key insights. First, different from CNN models, the self-attention mechanism of transformers enables the model to learn dependencies between hand joints and give more importance to the joints responsible for performing a specific task, e.g., pressing a key on the keyboard. As such, transformers can be highly effective in extracting key patterns from noisy handpose data and greatly boost attack robustness in noisy data settings.

Second, despite the common belief that transformers require vast amounts of training data, our work demonstrates their suitability for our attack, even with limited data. This is attributed to two design decisions. The first is that we exclusively train transformers on the current attack data, "forcing" them to concentrate solely on learning crucial features from this specific dataset. Next, we break down the main task into three sub-tasks or components, and train transformers separately for each component. This empowers each transformer to focus on its designated learning task. By adopting these design choices, we effectively leverage transformers to execute our attack, even

when faced with limited training data.

Attack Overview

Based on the above insights, we propose a new attack design, which applies a set of three sequential components to infer typed characters from noisy telemetry data. Our design tackles the challenging issue of noisy telemetry data through a two-step self-supervised learning approach, which is applied to each of the three components separately. By applying this self-learning method to each individual component, we design the attack to progressively extract and learn important keystroke information from the noisy input, while effectively reducing the spread of error in the end-to-end attack pipeline.

The three components. We break down the main task into three components: detecting keystroking telemetry frames, identifying pressing fingertips, and recognizing typed keys. The first component focuses on detecting telemetry frames[9] that represent keystroking events. The next component takes as input the detected keystroking frames and identifies the specific finger that touches the keyboard. The third component leverages the detected finger information to locate the pressing location on the keyboard and analyzes the pressing locations of all the keystroking frames to estimate the sequence of keys that the target user has entered.

Training per-component transformers. For each component, we first self-curate labels for its input telemetry data by performing statistical and motion analysis on the data. Here the statistical analysis focuses on a subset of hand joints closer to the keyboard and leverages general understandings of human typing behavior. After creating the initial labels for the telemetry data, we use them to train a deep neural network (DNN) model dedicated for this component. Note that this DNN model learns the detailed relationship between keystrokes and telemetry data from 3D coordinates of *all* hand joints (i.e., the full telemetry). To minimize the impact of the noise present in both the self-generated labels and the telemetry data, we employ multiple noise-aware training strategies during DNN model training.

9. Here each telemetry frame refers to the telemetry data collected on the target user at a specific time t.

Here we choose transformers as the DNN architecture for the first two components, but CNN for the last component due to the issue of class imbalance (details in §2.2.6).

End-to-end attack pipeline. Figure 2.13 plots the attack pipeline, which takes as input a sequence of 3D telemetry frames ($\mathbf{H}_{org}$) captured during the target's typing session, and applies the three sequential components to analyze the data and train DNN models. This pipeline produces three DNN models, one per component: (i) a transformer based keystroke detector $\mathcal{D}$, which identifies the set of telemetry frames where the target enters a key, (ii) a transformer based finger identification $\mathcal{F}$, which determines the finger used to press the key, and (iii) a CNN-based keystroke classifier $\mathcal{C}$, which predicts the key entered for each keystroking telemetry frame. The final inference process only employs $\mathcal{D}$ and $\mathcal{C}$, since $\mathcal{F}$ is only used to produce cleaner labels for the last component and train a more effective $\mathcal{C}$.

Supporting observed telemetry and rendered handpose attacks. To enable these attacks (i.e., using $\mathbf{T}_{3d}$ or $\mathbf{T}_{2d}$), we deploy a pre-processing module that transforms $\mathbf{T}_{3d}$ or $\mathbf{T}_{2d}$ into a coordinate system equivalent to that of $\mathbf{H}_{org}$. This is the "coordinate transformation" module in Figure 2.13.

2.2.6 Detailed Attack Design

We now present in detail each of the three sequential components of the attack pipeline, the pre-processing module, followed by the end-to-end attack implementation.

2.2.6.1 Detecting Keystroking Frames

The first component studies the input sequence of 3D telemetry frames to identify keystroking events and their associated telemetry frames. This is done in two steps, first applying statistical and motion analysis to self-label the frames,then using these labeled data to train a transformer-based DNN model, leveraging noise-aware model training techniques. Note that our discussion assumes the telemetry frame is from $\mathbf{H}_{org}$.

Step 1: Self-curating labels via motion analysis. When typing on a keyboard, each keystroking action is a characteristic movement of the finger, which starts from reaching the key, then a brief moment of zero acceleration as the finger hits the key, followed by a return to the starting position. This movement creates a distinctive pattern, represented as a positive peak in the plot of the second derivative of the keystroking finger's depth coordinate (or the y value of the fingertip in the telemetry data).

This general pattern only stands when the fingertip depth (y) values are measured at a very high accuracy (i.e., sub-mm precision). Today's VR hand tracking is far from that. The noise in the telemetry data, in combination with hesitant typing behavior, produces many "fake" positive peaks. On the other hand, these fake peaks often display a smaller amplitude than the real ones. Thus, we model the overall peak series as a mixture of two Gaussian distributions [261] and compute a decision threshold to produce an equal error rate between admitting a fake keystroke and missing an actual keystroke.

Step 2: Training a transformer-based keystroke detector. In Step 1, we analyze the unique feature of fingertip acceleration in the y axis to identify keystroke events, resulting in a set of "rough" labels for all telemetry frames. Using these labeled data, we train a *transformer*-based [238] keystroke detection model in Step 2. The choice of the transformer architecture deserves some explanation. The telemetry data consists of 3D positions of 42 hand joints in each frame, which have natural correlations in their movements and configurations across time and space. Furthermore, these joint configurations have spatial constraints due to the hinge nature of hand joints. To effectively learn and accurately predict finger activity, it is essential to have a model that understands these correlations and constraints. Along this line, the self-attention mechanism used by transformers [26] is well-suited for our learning task as it allows the model to learn dependencies between hand joints and give more importance to the joints relevant for performing a particular task (i.e., keystroke).

To model the spatial-temporal relationship among hand joints, we choose the DSTA-Net ar-

chitecture proposed by Shi et al. [208] for skeleton-based gesture recognition. This architecture calculates spatial attention to create a combined spatial representation of joints and then utilizes it to compute the temporal attention of joints across time. The pre-trained model provided by [208] and its training data only cover a single hand. Thus, we adapt the model architecture to support both hands and train the detection model from scratch to run a binary classification task (keystroke or non-keystroke). The input to the transformer-based classifier is a collection of 16 telemetry frames to capture the spatial-temporal movement of the hands. Specifically, we first identify each telemetry frame labeled as "keystroke" in Step 1, then take 7 telemetry frames before it and 8 telemetry frames after it to form a telemetry segment of 16 frames, and label it as "keystroke." We also build non-keystroke segments by grouping 16 non-keystroke telemetry frames between two detected keystroke events.

Noise-aware model training. Since both the telemetry data and the labels we self-curate (during Step 1) contain errors, we apply two noise-aware training techniques when training the transformer model. First, we apply mixup data augmentation [271] to prevent the model from overfitting into the training data and label. This is done by generating new training samples via blending two existing training samples using convex combinations, teaching the model to make linear predictions on those new data. The presence of noise in both the data and labels impedes the model's ability to learn these linear relationships and naturally prevents overfitting.

Second, we use bootstrapping for refurbishing labels which relies on the idea that as training progresses, the model learns to predict the correct class with higher confidence. Therefore, for a sample with a noisy label, the model's own prediction is a better source of the ground truth label and should be used as such during training whereas correctly labeled samples should be trained in the regular way. To identify the noisy labels, Arazo et al. [20] leverage the idea that noisy labels are learned in the later stages of training. Therefore, the difference in the loss can be used to identify the noisy labels which can then be trained using the model's predictions. In practice, convex combinations of the noisy and model-predicted labels are used. For samples that are noisy

with high probability, the model-predicted labels are weighted higher in the refurbished label and vice versa.

2.2.6.2 Detecting the Pressing Fingertip

Self-curating labels using finger displacement. After detecting keystroke events, it is important to identify the finger used to generate each keystroke in order to estimate the location of the keystroke. For this, we adopt the idea from [261] that given any position of the hand, the finger used to press the key needs to move the most in comparison to the non-pressing fingers. Therefore, we compute the displacement of each fingertip, at the moment of a keystroke, from its mean position across the entire typing session. We label the fingertip with the maximum displacement as the pressing finger. However, due to subtle typing behaviors and sensor noise, finger identification with this method is prone to errors thus some pressing fingertips are often incorrectly labeled.

Training a transformer for fingertip identification. Using the labeled fingertips, we train another transformer model to leverage features in the raw telemetry data. Again, this model is trained in a noise-aware manner from scratch on a set of 16 raw frames for each detected keystroke and predicts which finger was used to press a key. After training, this model is evaluated on the same data used for training in order to correctly predict the noisy labels. However, noisy training has its limits. If the proportion of noisy labels in the training set is beyond a certain threshold, noisy training techniques are unable to produce a reliable model due to overfitting. We identify these cases by observing the clustering output (§2.2.6.3) and avoid using the transformer in this component if it produces a worse clustering result.

2.2.6.3 Recognizing Typed Keys

Self-curating labels using clustering & HMM. After obtaining fingertip labels (from the fingertip transformer), we use the 3D coordinates of each identified fingertip, at the precise moment

of the keystroke, to estimate a touchpoint map which corresponds to the layout of the keyboard. Since keys are separated and have fixed locations on a keyboard, this form factor naturally clusters keystrokes generated from the same key and allows us to run unsupervised inference to reconstruct the typed content. We use K-Means clustering with K=38 to cluster the keystrokes and label these clusters arbitrarily. This labeling gives us a sequence of cluster IDs where each element in the sequence corresponds to a keystroke in the order that it occurred. The final task is to map each of these elements to one of 29 keys (period, comma, space key, and the 26 letters of the English alphabet). This is the kind of task Hidden Markov Models are well-suited to solve. We explain the detailed HMM implementation in Appendix 2.2.10.2. The output of the HMM gives us another set of labels which we use to train a DNN model for keystroke classification.

Due to the accumulated errors in our pipeline, there are errors in the HMM's prediction of the typed content. Therefore, we use the consistency checks between the clustering and the HMM's output introduced by [261] to identify the incorrectly recognized typed keys. As a result, instead of solely relying on noisy training, we can remove noisy labels from the self-curated labels to create a cleaner training dataset for the DNN-based keystroke classifier. The details of these consistency checks along with the explanation of other clustering related components are explained in Appendix 2.2.10.1.

Training a CNN for recognizing keys. The natural choice for training the keystroke classifier would be the transformer architecture used previously. However, after filtering, the training dataset (of 29 classes) is much smaller compared to those used to train the previous transformer models. This dataset is also severely class imbalanced due to the uneven distribution of the alphabet frequency in the English language. Due to these factors, a transformer-based keystroke classifier does not perform well.

Thus, we opt for a 3D-CNN model (ResNext-101 [255]) for keystroke classification. This model is pretrained on the Jester [146] dataset which contains videos of humans performing different gestures and can therefore be fine-tuned with limited data. However, our data is in the form

of 3D hand coordinates whereas the 3D-CNN is trained on images. Following common practice [207, 68], we convert our hand joint data to images by using the (x, z) coordinates as the pixel locations and the (x, y, z) coordinate as the pixel value. Like before, such a sample in the training dataset consists of 16 consecutive raw telemetry frames corresponding to a detected keystroke and its label obtained from the HMM.

2.2.6.4 Coordinate Transformation

The adversary with access to original telemetry data ($\mathbf{H}_{org}$) poses the strongest threat, as they have the "cleanest" data among all three attacks. For the other two attacks, $\mathbf{T}_{3d}(PoV)$ is a lossy, transformed version of $\mathbf{H}_{org}$, while $\mathbf{T}_{2d}(PoV)$ is a projected version of $\mathbf{T}_{3d}(PoV)$ with the depth dimension removed. Since the above described pipeline is designed using the 3D coordinate system of $\mathbf{H}_{org}$, we execute the other two attacks by first applying a coordinate transformation, which converts their attack data into an equivalent version of $\mathbf{H}_{org}$.

Converting $\mathbf{T}_{3d}(PoV)$ to $\mathbf{H}_{org}$. $\mathbf{T}_{3d}(PoV)$ is the telemetry observed from $\mathcal{A}$'s point of view. It is generated from $\mathbf{H}_{org}$ by first applying 3D transformation defined by PoV, including translation, scaling, and rotation [116]. This is followed by perspective projection [116, 35] where the telemetry joints are projected onto a 2D viewing window (like a camera frame) based on $\mathcal{A}$'s PoV. Together, this transformation and projection create misalignments and distortions that interfere with the motion analysis used by our attack pipeline.

To execute the observed telemetry attack, we apply a 3D rotation to $\mathbf{T}_{3d}(PoV)$ by calculating the dot product of each telemetry joint's coordinates[10] with standard rotation matrices [116]. This allows us to convert $\mathbf{T}_{3d}(PoV)$ into an $\mathbf{H}_{org}$ approximation, which can be input to our pipeline.

Multi-camera rendered handpose attack. $\mathbf{T}_{2d}(PoV)$ is a 2D version of $\mathbf{T}_{3d}(PoV)$ without the z (i.e., the depth information). Given the hefty information loss, the attack becomes ineffective

10. Since z is in a different unit than x and y, normalizing them into the same scale is needed before applying the 3D rotation.

if the adversary $\mathcal{A}$ has only a single $\mathbf{T}_{2d}(PoV_1)$. However, $\mathcal{A}$ can easily compensate for the information loss by adding another *hidden* camera at a different PoV_2 and collect $\mathbf{T}_{2d}(PoV_2)$, and use them to recover the lost depth values based on stereo epipolar geometry [76]. Setting up multiple, simultaneous, *hidden* cameras by a single VR user is a functionality supported by VR systems and developments. For instance, $\mathcal{A}$ could register multiple sybil accounts and place their avatars in the same VR environment with $\mathcal{U}$ to capture $\mathbf{T}_{2d}$ from different angles.

2.2.6.5 Putting It All Together

Once the DNN models are trained, we run inference using the DNN-based keystroke detector $\mathcal{D}$ and classifier $\mathcal{C}$. The keystroke detector identifies the telemetry frames which contain keystrokes and these are passed to the classifier to infer the key used for that keystroke. The finger identifying DNN becomes redundant in the inference stage as its purpose is to obtain a cleaner clustering that improves the labels in the training set of the keystroke classifier. The keystroke classifier sidesteps the issue of finger identification by directly inferring the keystroke labels using patterns in raw telemetry frames.

After reconstructing the typed content, we further improve it using spell check. We find that conventional spell checking tools operate at a word level whereas spell checking can benefit from context. Thus, we use the spelling and grammar checking tool available in Google Docs, which makes use of the context using machine learning techniques [92], to improve the recovered text. We build an automated version of the Google checker using the Selenium [**?**] library.

2.2.7 Evaluation

We assess the effectiveness of the proposed attack through user studies approved by our Institutional Review Board (IRB21-1396). We evaluate five key aspects of the attack:

- **Effectiveness of original telemetry, observed telemetry, and rendered handpose attacks (§??)**, which utilize three types of telemetry data;

- **Performance under different VR configurations (§2.2.7.3)**, where we vary the distance between the avatars, the keyboard device, and the content type;
- **Attack effectiveness on 15 users** of varying typing styles (§2.2.7.4);
- **Comparison to other solutions (§2.2.7.5)**, including the baseline solutions (attack by transferability and unsupervised inference) discussed in §2.2.4.2 and the attack design of [261];
- **Attack complexity (§2.2.7.6)** in terms of computation time on a standard server and a breakdown across components.

2.2.7.1 Experimental Setup

User (or target) configuration. By default, each of our study participants wears a Meta Quest Pro VR headset and sits in front of a table (on a wheeled chair). A physical keyboard (Logitech K375s) is placed on top of the table and connects to the headset via Bluetooth. Before each typing experiment, we leave sufficient time so that our participants can freely adjust their headset, the chair, and the keyboard. On average, each typing session lasts about 25 minutes.

While wearing the VR headset, each participant types on the keyboard. Through the headset, the participant can see a rendered (virtual) display showing the text they have entered and those to be typed. They can also see their virtual hands, rendered locally using the tracked telemetry $\mathbf{H}_{org}$. By default, we ask each participant to type, on average, 26 email sentences randomly selected from a popular corporate email dataset [52], and to correct their typing mistakes using the backspace key. To simulate actual working conditions, each participant is allowed (and encouraged) to take breaks during the typing session, including pausing, leaving, and returning to their seat. Both the table and the keyboard remain stationary during a typing session.

Attacker configuration. We consider an attacker who is able to acquire the telemetry data of the target. The telemetry data is in one of the three forms: original telemetry $\mathbf{H}_{org}$, observed telemetry $\mathbf{T}_{3d}(PoV)$, and rendered handpose $\mathbf{T}_{2d}(PoV)$. The sampling rate of telemetry is around 60Hz, a

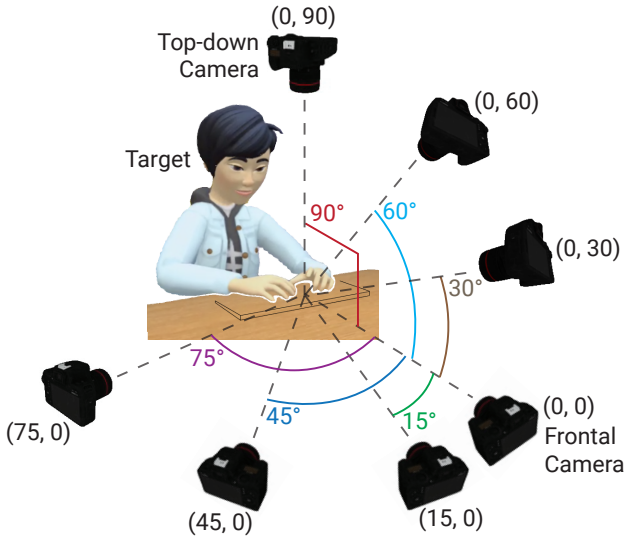

Figure 2.14: A graphical illustration of virtual camera PoVs. $(0, 0)$ means a frontal camera at the same height as the keyboard pointing towards the target's typing hands. (x, y) means the camera is horizontally, vertically deviates by x, y degrees from $(0, 0)$, e.g., $(0, 90)$ represents a top-down camera. Credit: The target avatar is created using the built-in avatar tool on a Meta Quest Pro headset.

standard setting used by both Quest 2 and Pro VR headsets. The latter two attacks depend on the attacker parameter PoV, i.e., the virtual camera placement configured by the attacker. Figure 2.14 illustrates the PoV configuration. $PoV = (0, 0)$ means the attacker places a frontal virtual camera at the same height as the keyboard, which points toward the target's typing hands. $PoV = (x, y)$ means the virtual camera deviates vertically and horizontally from $(0, 0)$ by x, y degrees, respectively. For all PoV configurations, the virtual camera always points toward the target's typing hands.

Dataset. We build a Unity program to collect telemetry data used by our attack and ground truth typing input for our evaluation. For each user, we collect their telemetry data over a typing session (≈500 words). Each session's data, i.e., a set of telemetry frames, is used to implement the attack for the current user and the current session. Specifically, we first self-curate binary labels (i.e., keystroking or non-keystroking) for a subset of the telemetry frames, and use them to train a keystroke detector $\mathcal{D}$. We feed the entire session's data to $\mathcal{D}$ to identify the set of keystroking frames. Next, for all keystroking frames, we curate finger-specific labels to train a finger identifier

$\mathcal{F}$ and apply $\mathcal{F}$ on all keystroking frames to help curate the key-specific labels, followed by a consistency check filter to construct the labels for a subset of keystroking frames. The resulting labeled data is used to train the keystroke classifier $\mathcal{C}$, which is then applied to all keystroking frames to produce the attack output.

Here we note that the quantity of training data used to train the three DNNs ($\mathcal{D}$, $\mathcal{F}$, $\mathcal{C}$) differs across users and sessions. As reference, Table 2.18 in the Appendix displays, for each of the 15 users, the quantity of telemetry frames used to train these DNNs and the quantity of inference data being fed into the trained DNNs.

Evaluation metrics. Following an existing work [261], we evaluate each attack attempt by comparing the typed and inferred content at the character, word, and semantic levels.

- Character error rate (CER): It measures errors at the character level by computing the ratio of the number of errors and the length of the actual typed content. The number of errors is measured by the minimum edit distance, comprising insertions, deletions, and substitutions, required to convert the inferred content to the typed content. The lower the CER, the higher the accuracy of the inferred content.

- Word error rate (WER): The WER is computed using the same ratio as the CER with the difference that the edit distance is computed at the word level. It is worth mentioning that while WER penalizes words that are not exact matches, they may be comprehendible by a human reader. We use the WER function available within the Hugging Face [70] library as it uses dynamic string alignment to best match word indices for comparison.

- Content similarity using plagiarism detection: Plagiarism detectors are used to compute similarity between documents and are thus a natural fit for evaluating the recovered content. We use the plagiarism detector offered by CopyLeaks [54] which provides a similarity score in the range of 0-100% for the actual and inferred content. With moderate errors in the inferred content, the similarity score operates as a hybrid between character and word level

similarity. However, with higher levels of errors, the similarity score usually collapses and is much lower than the word recovery rate (1 - WER). We refer to the obtained similarity score as the **similarity**.

2.2.7.2 Attack Effectiveness vs. Telemetry Data

We begin by examining the attack performance on original telemetry, observed telemetry, and rendered handpose data. It is intuitive to assume that the original telemetry attack is the strongest as the adversary has access to $\mathbf{H}_{org}$, the "cleanest" data among the three. As explained earlier in §2.2.6.5, the observed telemetry data ($\mathbf{T}_{3d}(PoV)$) is produced from $\mathbf{H}_{org}$ using a lossy, non-linear perspective projection, while the rendered handpose data ($\mathbf{T}_{2d}(PoV)$) suffers even more information loss from the 3D-to-2D projection.

In Table 2.11, we compare the performance of the three attacks, by varying the attacker's virtual camera PoV. For consistency, we invited a single participant as the attack target for all the experiments. We also include the result for the **two-camera rendered handpose attack**, where the attacker sets up two virtual cameras to collect two $\mathbf{T}_{2d}$ at different PoVs.

Effectiveness of original telemetry attack. Our results show that the Original telemetry attack (using $\mathbf{H}_{org}$) is highly successful – the attacker effectively recovers 85.8% of the words in the typed text. The recovered text has a semantic similarity of 87.4% to the original.

Observed telemetry attack with different PoVs. In the observed telemetry attack using $\mathbf{T}_{3d}(PoV)$, the attacker needs to set up a virtual camera at an observation point PoV to collect $\mathbf{T}_{3d}(PoV)$. We experiment with 7 different PoV values and find that the attack is consistently effective – the attackers can recover around 78% of the typed words. The performance is slightly less than that of the original telemetry attack because the inverse transformation of $\mathbf{T}_{3d}(PoV)$ produces a noisy version of $\mathbf{H}_{org}$. We observe the same performance trend from other participants (see Table 2.17 in Appendix).

Two-camera rendered handpose attack. Using a single virtual camera, the rendered handpose

Attack Type	Attack Data	Attacker's PoV	CER (%)	WER (%)	Sim. (%)
Original Telemetry	$\mathbf{H}_{org}$	–	4.7	14.2	87.4
Observed Telemetry	$\mathbf{T}_{3d}(PoV)$	$(0,0)$	6.6	21.6	80.4
		$(0,30)$	6.2	21.4	83.7
		$(0,60)$	6.5	21.4	81.5
		$(0,90)$	7.2	23.8	74.6
		$(15,0)$	5.8	18.0	86.1
		$(45,0)$	6.2	19.4	84.2
		$(75,0)$	7.6	21.0	77.6
Single camera Rendered Handpose	$\mathbf{T}_{2d}(PoV)$	$(0,0)$	29.3	78.0	0.0
		$(0,45)$	23.7	58.6	2.9
		$(15,0)$	28.5	66.8	0.0
Two-camera Rendered Handpose	$\mathbf{T}_{2d}(0,0)$ & $\mathbf{T}_{2d}(PoV)$	$(0,15)$	6.9	20.8	85.3
		$(0,45)$	6.3	20.6	81.7
		$(0,75)$	6.5	20.0	80.5
		$(15,0)$	7.6	21.2	79.0
		$(30,0)$	5.0	17.4	84.1

Table 2.11: Performance of original telemetry, observed telemetry and rendered handpose attacks, using $\mathbf{H}_{org}$, $\mathbf{T}_{3d}$, and $\mathbf{T}_{2d}$ handposes, respectively. We also include an extended rendered handpose attack using two $\mathbf{T}_{2d}$ at different camera PoVs.

attack (using $\mathbf{T}_{2d}(PoV)$) is ineffective because $\mathbf{T}_{2d}(PoV)$ is a noisy 2D projection of $\mathbf{H}_{org}$, making it hard to estimate the 3D telemetry. Yet we can overcome this challenge by setting up another virtual camera at a sufficiently different PoV. With two different $\mathbf{T}_{2d}$ observations, we can estimate the 3D telemetry $\mathbf{H}_{org}$ based on the stereo epipolar geometry [76]. In our experiments, we complement a frontal-view camera $\mathbf{T}_{2d}(0,0)$ with either a side-view or a vertical-view camera. Results in Table 2.11 show that the two-camera rendered handpose attack performs comparably to the observed telemetry attack. Again, these observations are confirmed across other users (see Table 2.17 in Appendix).

Real-world implications. The above results demonstrate the effectiveness of the three attacks. The success of the two-camera rendered handpose attack is particularly alarming due to two reasons. First, the attack is simple and stealthy. To collect $\mathbf{T}_{2d}(PoV)$, the attacker $\mathcal{A}$ does not need to hack any server or VR headset, but just acts as a benign user in the same VR environment with $\mathcal{U}$ and performs screen recordings of their VR screen on the headset. After applying video-based

hand tracking[11] on the recorded video of the VR view, $\mathcal{U}$ obtains $\mathbf{T}_{2d}(PoV)$. Second, $\mathcal{A}$ can register multiple sybil accounts in VR and record $\mathcal{U}$'s typing from many PoVs. With more versions of $\mathbf{T}_{2d}$, $\mathcal{A}$ can potentially obtain better estimates of the original telemetry $\mathbf{H}_{org}$ and improve attack effectiveness.

2.2.7.3 Effectiveness under Different VR Settings

To test the generalizability of our attack, we conduct multiple experiments by varying factors involved in our end-to-end attack pipeline. These include changing the (virtual) distance between the adversary's virtual camera to the target's avatar, the physical keyboard type, the typed content, and the amount of data available to launch the attack. To maintain consistency, a single participant in our user study was the target in all these attack scenarios.

Varying avatar distance. To assess the impact of avatar distance in the virtual world, we conducted an experiment in which four adversaries were positioned at increasing distances (0.6m, 1.2m, 1.8m, and 2.4m) from the target avatar during a single typing session. The adversaries recorded four separate sequences of transformed 3D telemetry ($\mathbf{T}_{3d}(0,0)$) using their virtual cameras. The results are summarized in Table 2.12, where we consider 0.6m attack as the baseline with CER $= 3.8\%$ and report the CER difference between the 1.2m/1.8m/2.4m attack and the baseline. The results confirm that the virtual distance between the target and the adversary avatars does not impact the attack performance.

Avatar distance (m)	**1.2**	**1.8**	**2.4**
CER $-$ CER$_{0.6}$ (%)	-0.17	-0.06	$+0.03$

Table 2.12: Attack performance when the virtual distance between the target and the adversary avatars varies. The default attack is conducted with the avatar distance $=$ 0.6m. We report the CER difference from that of the default attack (CER $= 3.8\%$).

Varying keyboard type and size. People can use different keyboards of various sizes. To evaluate our attack under such conditions, we ask our participant to type on three different keyboards:

11. Open source hand tracking tools are commonly available online, e.g., MediaPipe [237] and OpenPose [214].

Logitech K375s, Logitech MX Keys Mini, and Apple Magic keyboard. The results for these experiments, displayed in Table 2.13, show that our attack is successful against all three physical keyboards.

Keyboard	**Dimension** (cm)	**CER** (%)	**WER** (%)	**Similarity** (%)
Logitech K375	42.7 × 13.0	4.7	14.2	87.4
Logitech MX	29.3 × 12.8	6.2	19.6	86.4
Apple Magic	28.0 × 11.5	6.6	17.9	82.4

Table 2.13: Attack performance when the target types on three different physical keyboards.

Varying content type and length. We investigated the performance of our attack with different types and lengths of text content. In addition to corporate emails, a participant typed approximately 500 words of text from scientific papers and medical patents, which contain technical terms and uncommon words. To evaluate the effect of limited typing data, we also conducted the attack when the participant typed only around 250 words. The results, presented in Table 2.14, indicate that our attack is effective with various content types and lengths tested.

	Word Count	**CER** (%)	**WER** (%)	**Similarity** (%)
Abstract	509	5.2	18.1	83.3
Patent	501	4.6	20.8	80.6
Emails	501	4.7	14.2	87.4
	262	5.4	17.6	84.5
w. numbers	488	5.5	18.7	81.9

Table 2.14: Attack performance when the target types content of different kinds and lengths.

Impact of numbers in text. The Enron [52] dataset includes sentences that contain numbers, which account for 1.2% of keystrokes. Using these sentences, we evaluate our attack to study the impact of number keys. Intuitively, the presence of numbers in the typed content would affect the HMM-based labeling process because HMM (trained on text) is unable to recognize numbers and will label them as letter keys. However, because these number keys are physically separated from letter keys on the keyboard, they will be filtered out by our consistency checks and will not appear in the training data for the keystroke classifier. As such, the moderate presence of numbers has

Approach	P1			P2			P3			P4			P5			P6			P7		
	CER (%)	WER (%)	Sim. (%)	CER (%)	WER (%)	Sim. (%)	CER (%)	WER (%)	Sim. (%)	CER (%)	WER (%)	Sim. (%)	CER (%)	WER (%)	Sim. (%)	CER (%)	WER (%)	Sim. (%)	CER (%)	WER (%)	Sim. (%)
Our Attack	**1.2**	**4.8**	**97.6**	**4.7**	**11.9**	**93.5**	**4.7**	**14.2**	**87.4**	**4.8**	**13.8**	**88.1**	**5.0**	**15.6**	**83.6**	**7.1**	**18.2**	**81.9**	**7.3**	**18.8**	**84.2**
Stats. Analysis	6.7	22.0	82.0	20.0	51.0	25.1	18.3	51.8	6.2	22.4	52.5	7.4	19.3	51.4	19.1	17.9	46.3	6.9	13.2	34.2	56.8
Transferability	77.8	99.6	0.0	57.8	100.0	0.0	53.4	100.0	0.0	47.4	86.2	0.0	61.8	96.1	0.0	31.9	87.2	0.0	49.9	92.8	0.0
Attack in [261]	17.6	56.5	8.3	33.1	82.1	0.0	8.0	26.4	65.8	16.8	52.9	22.7	17.5	48.1	14.7	44.4	97.6	0.0	48.0	100.0	0.0

P8			P9			P10			P11			P12			P13			P14			P15		
CER (%)	WER (%)	Sim. (%)	CER (%)	WER (%)	Sim. (%)	CER (%)	WER (%)	Sim. (%)	CER (%)	WER (%)	Sim. (%)	CER (%)	WER (%)	Sim. (%)	CER (%)	WER (%)	Sim. (%)	CER (%)	WER (%)	Sim. (%)	CER (%)	WER (%)	Sim. (%)
8.4	**21.7**	**80.5**	**8.8**	**25.8**	**77.7**	**9.5**	**25.8**	**77.7**	**12.5**	**34.5**	**63.9**	**13.3**	**33.9**	**50.7**	**13.3**	**34.1**	**51.1**	**14.2**	**44.6**	**34.4**	**15.9**	**44.4**	**19.7**
28.1	62.8	0.0	27.6	57.7	0.0	24.0	61.4	4.1	29.1	70.1	0.0	31.7	70.3	0.0	45.0	91.6	0.0	38.6	83.4	0.0	54.5	96.8	0.0
44.3	100.0	0.0	47.7	90.6	0.0	47.7	85.7	0.0	54.3	94.3	0.0	20.1	51.0	25.3	91.0	100.0	0.0	14.3	33.9	59.3	47.9	94.0	0.0
32.7	78.9	0.0	30.5	64.3	0.0	14.9	37.2	50.6	16.2	41.9	36.7	26.6	67.5	0.0	20.0	53.6	5.5	31.6	91.0	0.0	62.4	99.2	0.0

Table 2.15: Attack performance for all 15 participants (P1-15).

minimum impact on our attack. Our results in Table 2.14 (last row) also confirm this hypothesis.

2.2.7.4 Performance across Users

To assess the effect of individual typing styles on the proposed attack, we recruited 15 participants to our user study (age: 21-53, 7 males and 8 females). In Table 2.15, we report the performance of our original telemetry ($\mathbf{H}_{org}$) attack against each participant. Overall, the attack is highly effective – for 13 out of 15 participants, our attack accurately recognizes 86%-98% of typed keys, and the recovered text retains 50%-98% of the original content's meaning.

The successful cases (P1-P13). After closely observing these users, we identified two distinct typing patterns that increase users' vulnerability to our attack. First, they type with prominent motion, so the hand tracking sensors on the VR headset can capture sufficient amounts of finger movements. The hand tracking accuracy is hardware dependent and does vary across users. Second, their body posture and hand orientation minimize self-occlusion across their fingers, meaning the joints are not blocking each other from the view of the hand tracker (located on their VR headset). When a joint is obscured, the joint's coordinates are often inaccurately estimated, leading to random tracking noise.

The unsuccessful case (P14 and P15). Our attack is less effective on P14 and P15, due to high hand tracking noise. For P14, the VR hand tracker is unable to track the pressing fingertip location

(i.e., the x and z values) accurately. To illustrate this, Figure 2.15 compares P14's keystroke pressing locations reported by the VR tracker to those of P1. We see that P14 has many more overlapping keystrokes than P1, making it hard to create reliable labels.

For P15, the major issue is the height data (y) collected by the VR hand tracker. This is because, rather than having the keystroking finger much closer to the keyboard than the other fingers, P15's fingers stay at roughly the same height level when pressing keys. As a result, the tracking data is particularly noisy on the y axis, making it hard to reliably detect keystroke events and the pressing finger.

2.2.7.5 Comparison to Other Solutions

We compare our attack design to baseline solutions discussed in §2.2.4.2 and related work [261] discussed in §2.2.5.1. Overall, our attack design significantly outperforms these solutions.

Our attack design vs. baselines. We compare our approach with two baseline approaches discussed in §2.2.4.2. The first baseline is *unsupervised inference via statistical analysis*, which takes three sequential steps of detecting keystrokes, identifying pressing fingertips, and recognizing keys. These are the same three steps used by our attack design to generate initial labels but this baseline does not involve any DNN models. The second baseline is *attack by transferability*, where for each target, we use ground truth labels on all 14, non-target participants to train a DNN keystroke classifier, and hope it can also recognize the target's keystrokes. Note that in this implementation of transferability-based attacks, we assume perfect keystroke detection on the target user. Thus, the result represents the best case of transferability-based attacks.

The results in Table 2.15 show that our attack design significantly outperforms the two baselines. Compared to the *unsupervised inference via statistical analysis*, our approach curates labeled training data for training transformer-based DNN models, which can better capture the intrinsic features of the noisy training samples. The second baseline *attack by transferability* failed largely because human typing behaviors are highly user-specific and do not transfer to others [261]. Our

approach overcomes this challenge by utilizing the target's own data to learn their specific typing behaviors.

Our attack design vs. [261]. We obtain the source code from the authors of [261] and follow the training method described in [261]. To train and test the CNN classifiers used in [261], we convert the telemetry frames to images. For a fair comparison, we apply the same conversion method used by our approach (see §2.2.6.3).

The results in Table 2.15 show that our attack design significantly outperforms that of [261]. As discussed in §2.2.5.1, the amount of noise in our attack data is significantly higher than that of [261], rendering the method of [261] ineffective. Our design addresses this challenge by deploying multiple task-specific transformers where the self-attention mechanism allows these models to extract key patterns from noisy data.

2.2.7.6 Attack Complexity

We measure attack complexity by the amount of time required to produce the recovered text after the typing session stops. We run our end-to-end attack pipeline on a standard server with an AMD EPYC 7313P 16-Core processor, and train all the models on a single NVIDIA A40 GPU. The overall runtime of our attack is 423 minutes for a 500-word, 19-minute typing session. The majority of time was spent on the training of transformer-based keystroke detector (79 mins), the finger identifier (64 mins), and the CNN-based keystroke classifier (259 mins). The other components only take 21 minutes in total. Specifically, clustering & HMM take 3 minutes and spell checking takes 11 minutes.

2.2.8 Defenses

Our study shows that by either intercepting the telemetry data used to render a target's avatar or simply observing rendered hand movements of their avatar, an attacker in the same virtual environment can successfully recover the text physically typed by the VR user. Untreated, these

attacks can cause significant damage to users. Given this threat, it is important for VR platform and app developers to exercise caution and implement adequate safeguards during the development process. We discuss several defense options below and their impact on the attacks and the VR experience.

Defense 1: limiting access to telemetry (hand tracking) data. The first type of defense seeks to minimize the chance of leaking sensitive handpose data ($\mathbf{H}_{org}$ and $\mathbf{T}_{3d}$) to attackers. After detecting the user is typing, the VR system can either ban access to its hand tracking API by any application, put a significant limit on the query frequency, or largely reduce the sampling rate of the hand tracking module. For example, by reducing the sampling rate from the default 60fps to 15fps, the defense can increase attack WER from 14.2% to 38%. When further reduced to 6fps, the attack becomes ineffective.

Defense 2: adding noise to telemetry data. Another type of defense is to add noise to the 3D hand tracking data captured by the wearer's headset (i.e., $\mathbf{H}_{org}$). In particular, by perturbing the depth (y) value of $\mathbf{H}_{org}$, one can effectively confuse the attacker at the stages of keystroke detection and finger identification, preventing them from identifying the keystroke events and/or the pressing finger. In our design, we choose to add zero-mean Gaussian noise to the y value. This is because our empirical measurements show that the noise naturally present in the hand tracking data follows a Gaussian distribution. Adding zero-mean Gaussian noise on top makes it hard for the attacker to denoise the data.

Table 2.16 summarizes the attack performance on the original telemetry data ($\mathbf{H}_{org}$) before and after applying this defense. We consider two noise levels: *moderate*, where we add zero-mean Gaussian noise with an STD of 0.3 $\times$ key width, and *high*, where the STD rises to 0.5 $\times$ key width. We see that adding a moderate level of noise can already disrupt the attack, raising CER from 4.7% to 20.1%, WER from 14.2% to 55.4%, and dropping semantic similarity to 14%. Under a high level of noise, WER further rises to 71.4% and semantic similarity drops to 0%. We also visually inspect the target's avatar under both noise levels. The avatar still displays normal typing behavior under

moderate noise, while the typing speed appears to be considerably accelerated under high noise.

Noise Level (STD)	**CER** (%)	**WER** (%)	**Similarity** (%)
No noise	4.7	14.2	87.4
Moderate noise: 0.3 × Key Width	20.1	55.4	14.1
High noise: 0.5 × Key Width	26.6	71.4	0.0

Table 2.16: Attack performance after perturbing the depth axis of $\mathbf{H}_{org}$ using zero-mean Gaussian noise.

Impact on VR immersiveness. As expected, both defenses could impact the VR experience, because the user's physical hand motion is no longer synchronized with their avatar. Here the user can choose to run the defense directly on telemetry data collected by the VR headset, or only when the collected telemetry data leaves the VR headset. The first option provides a more thorough protection but affects how the target interacts with the VR environment. In particular, the target user may not see their correct typing movements inside the VR. The second option only affects the avatar observed by other VR users, but cannot resist attacks that can access the headset's telemetry data. Clearly, this represents an inherent tradeoff between usability and security.

2.2.9 Conclusion

Our study is the first to explore the possibility of keystroke inference attacks in a shared virtual reality environment. In this scenario, an adversary VR user can reconstruct text typed by another user by observing their avatar. Unlike prior work, our attack does not require physical observation of the target, only a noisy VR representation of their avatar's hand movements. This avatar-based attack highlights the tension between immersive experiences and personal/information security in VR. While adding noise or reducing tracking frequency can reduce the attack's effectiveness, it also impacts the immersive experience to varying degrees. This raises the important question of how to design VR systems that balance the user's desire for immersive and engaging experiences with the need to protect their personal and information security.

Limitations and Future Work. The major limitation of our attack pipeline is that we ultimately choose a 3D-CNN model for our keystroke recognizer because the self-labeled training data for this component is very limited and imbalanced. The 3D-CNN works adequately with limited training data as it has already been pretrained on a large gesture recognition dataset [146]. We were not able to find pretrained transformer models or large datasets with telemetry data for both hands. Nevertheless, as transformers are being successfully applied to non-NLP domains, it is reasonable to expect that pretrained models in the future would make our attack more effective.

2.2.10 Appendix

2.2.10.1 Clustering

Detecting and clustering thumb generated keystrokes. We observe that most users have very subtle motions with regards to typing keys with their thumbs (usually the space key). Therefore, our motion and displacement based techniques do not work well for thumb tip data. However, we are still able to detect thumb-based keystrokes via the movement of the other four fingers, because they have a distinct movement pattern when the thumb is used to press a key. Specifically, the four fingers move in tandem much more as compared to when a non-thumb finger is used for a keystroke. So if the fingers move together, the variance in their relative displacement is low and we use this signal to mark detected keystrokes as being thumb-generated.

However, this process is not very accurate and we follow the approach laid out by [261] to deal with thumb generated keystrokes. Despite having some confidence that a keystroke was thumb-generated, we treat it as if it was a non-thumb keystroke i.e., we proceed by including it in the clustering process by using the coordinates of the non-thumb key identified as having generated that keystroke. The only difference is that we do not let these keystrokes be clustered in the regular non-thumb key clusters. Rather, we let these keys form their own clusters and let the HMM decide which of these clusters correspond to thumb keystrokes, i.e., space keys and which are non-thumb

keystrokes.

Labeling consistency within each cluster. If the clustering was perfectly clean, we would expect each cluster to contain touchpoints of the same key and the HMM would most often label the clusters as such. However, given the errors in our clustering, the HMM often assigns different labels to keystrokes within the same cluster as it is not bound to assign a one-to-one mapping between a cluster's ID and its associated label. Rather, it relies on its learned parameters to decide the most likely labels at each time step. While the HMM can often assign the correct label to an incorrectly clustered keystroke, we find that minority labels within a cluster are unreliable. Therefore, we only select labels from the most dominant class in each cluster.

Cross-cluster label compatibility. By inspecting the majority label assigned by the HMM to the keystrokes in each cluster, we are further able to identify errors by previous modules in our pipeline. Specifically, we can identify errors in finger identification and the presence of non-keystroke events in our pipeline by doing a cross-cluster label consistency check. Intuitively, if two clusters have the same majority label, they should be close together in the clustering space since the clustering is spatial and each key has a fixed location on the keyboard. Therefore, if two clusters are far apart and have been assigned the same majority label, one of them cannot be legitimate. Following this observation, we sort such clusters by size and remove all but the largest one from our training set for the keystroke classifier.

Detecting backspaces For accurate keystroke inference, it is imperative that we detect backspaces with high precision as failure to do so leads to high rates of error accumulation. We use two heuristics to identify backspaces among the set of detected keystrokes. We assume that the backspace is one of the edge keys on the keyboard and secondly, it is often pressed multiple times together to fix typos. As a result, we label the corner cluster on our touchpoint map with the highest ratio of consecutive keystrokes associated with it. For every keystroke in the backspace cluster, we remove the previous non-backspace detected keystroke for it. After this step, the backspace cluster is not used anymore in the attack pipeline.

2.2.10.2 Hidden Markov Models (HMM)

The output of the clustering module gives us a sequence of cluster IDs where each element in the sequence corresponds to a keystroke in the order that it occurred. We want to map each element in the sequence to 29 keys (period, comma, space key, and the 26 letters of the English alphabet) and a sequence tagging algorithm such as the HMM can perform this mapping. We are interested in obtaining the actual keys being typed by the victim but are unable to directly observe them. However, we have obtained a sequence of cluster IDs that are causally related to the actual keys being typed. HMMs can model this causal relationship between the observed and unobserved events and then infer the unobserved events.

Formally, HMMs are characterized by two sets of parameters **A** and called the Transition and Emission matrices respectively. Matrix **A** models the probability of moving from one hidden state to another whereas the matrix **B** models the probabilities of hidden states producing the observed states. In our case, **A** models the actual $N=29$ keys and is a $N\ X\ N$ matrix whereas **B** models the $M < 50$ cluster IDs being produced by the alphabets and is therefore a $N\ X\ M$ matrix. Probabilities in both matrices satisfy the Markov assumption. Inspired by [261], we compute **A** by empirically estimating the transition probabilities in the English language by using 40,000 sentences from the CNN/DailyMail dataset [87]. Matrix **B** is learned using a variant of the Expectation-Maximization (E-M) [30] algorithm given **A** and the cluster sequence. Given **A, B**, and the cluster sequence, the HMM infers the most probable hidden state sequence, i.e., the typed keys.

However, since the E-M algorithm performs local optimization [100] it often converges to local minimas. To deal with this, we run multiple iterations of the HMM and select the model that produces the most number of high-confidence predicted labels using the consistency checks explained in Appendix 2.2.10.1.

Attack Data	Attacker's PoV	P1 CER (%)	P1 WER (%)	P1 Sim. (%)	P2 CER (%)	P2 WER (%)	P2 Sim. (%)
$\mathbf{H}_{org}$	–	1.2	4.8	97.6	9.5	25.8	77.7
$\mathbf{T}_{3d}(PoV)$	$(0, 0)$	2.1	7.2	95.6	10.3	29.1	75.5
	$(0, 45)$	1.7	5.0	97.8	9.9	27.9	67.3
	$(45, 0)$	1.6	6.6	95.8	9.4	25.0	70.3
$\mathbf{T}_{2d}(PoV)$	$(0, 0)$	56.9	96.8	0.0	32.5	77.9	0.0
	$(0, 45)$	10.2	26.1	74.4	41.7	93.8	0.0
$\mathbf{T}_{2d}(0, 0)$ & $\mathbf{T}_{2d}(PoV)$	$(0, 45)$	1.9	7.4	96.4	10.0	26.9	69.9
	$(15, 0)$	1.6	5.2	96.8	9.3	25.4	72.7

Table 2.17: Attack performance on two other participants' telemetry data. We include $\mathbf{H}_{org}$, $\mathbf{T}_{3d}$, $\mathbf{T}_{2d}$, and an extended rendered handpose attack using two $\mathbf{T}_{2d}$ at different camera PoVs.

Figure 2.15: P1 and P14's keystroke pressing locations tracked by the VR headset. The locations are based on ground truth keystroke timestamps and pressing fingertips.

	Keystroke Detector $\mathcal{D}$		**Keystroke Classifier $\mathcal{C}$**	
	Training	**Inference**	**Training**	**Inference**
	keystroking frames	*all frames*	*keystroking frames*	
P1	2925	181394	1647	2868
P2	2677	184139	1540	2849
P3	3070	50716	1475	2903
P4	3009	77494	1576	2950
P5	3096	70017	1351	3027
P6	3034	215462	1248	2916
P7	3235	184145	1676	2894
P8	3381	126165	1405	2887
P9	3023	121333	1257	2839
P10	3108	76549	1338	3026
P11	3078	79432	1341	2866
P12	3200	65410	1412	2950
P13	3194	37902	1233	2858
P14	3078	47422	1369	3042
P15	3274	36733	1333	2896

Table 2.18: For each of the 15 participants, the amount of training data, in terms of number of telemetry frames, used to train the DNNs (i.e., the keystroke detector $\mathcal{D}$ and the keystroke classifier $\mathcal{C}$), and the amount of inference data being fed into the trained models. When training $\mathcal{D}$, we set the self-labeled keystroking frames as positive samples in the training dataset, and extract the same amount of non-keystroking frames as the negative samples in the training dataset.

CHAPTER 3

MANIPULATING INTERACTIONS IN VIRTUAL REALITY SYSTEMS

3.1 Introduction

Recent advances in virtual reality (VR) are poised to change the way we interact with the world and each other [152, 101, 188, 38, 226, 73, 216, 123]. Today's VR headsets are offering an increasingly immersive experience comparable to reality itself, eliminating geographical barriers and facilitating social and workplace interactions through the use of personalized avatars [245, 77, 161, 156].

The flip side of such immersive interface is that when misused, VR systems can facilitate security attacks with far more severe consequences than traditional attacks [25]. By design, VR's sensory channels create a strong sense of realism and presence, fostering an implicit trust on the experience [200]. Attackers can exploit this trust to manipulate user experiences, leadings users to undoubtedly accept and respond to what they see and hear inside VR, whether it is scenes, movements, or conversations. For example, consider the following scenarios:

Scenario 1. *Alice spent the past two days taking her required employee training through her employer's VR app. Today, she attempts the final test using the same app. But despite her hard work, she keeps failing. Frustrated and discouraged, Alice eventually gives up and starts to doubt whether she is a good fit for the job. Elsewhere, her coworker Carl closes his laptop with satisfaction.*

Scenario 2. *Alice takes off her headset in frustration, after the latest argument with her partner Madison during their regular VR date night. Madison had been increasingly distant all week and actually ended their relationship tonight. Elsewhere, Carl takes off his headset and watches as a confused Madison storms off in VR.*

Both scenarios described above result from the same attack, where an attacker ("Carl") compromises the integrity of the user's VR system, and inserts an attack layer between the user and their VR interface. Under this attack, the user ("Alice") is no longer interacting directly with their

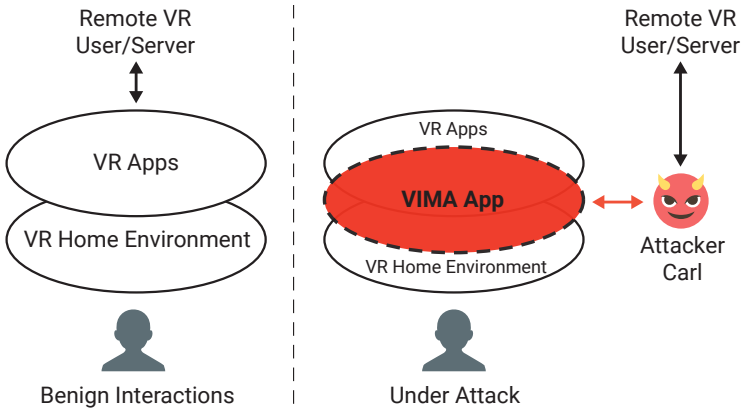

Figure 3.1: Our proposed `VIMA` attack: The attacker controls and manipulates the user's interaction with their VR system, by trapping the user inside a single, malicious VR app (the `VIMA` app) that masquerades as the full VR system.

expected VR counterparts (e.g., "CompanyX's server," "Madison"). Instead, they operate on (and stay trapped in) the attacker's VR app, interacting with their VR counterparts through a hidden layer of indirection. These interactions are being monitored, recorded, and modified in real-time by Carl. We name this new VR Interaction Manipulation Attack the `VIMA` attack and provide a high-level illustration in Figure 3.1.

The `VIMA` attack is a key threat to VR systems because of the following two properties. First, a `VIMA` attacker can take near total control of the user's VR experience. This includes silently observing the user's actions, recording input, passwords, or other sensitive data, but also hijacking and replacing their social interactions on the fly. In the aforementioned Scenario 2, Alice is a target of a `VIMA` attack and her VR date night with Madison is hijacked. Now both Alice and Madison only see/hear what the attacker wants them to see/hear. Ultimately, this attack can be enhanced by integrating advanced generative AI tools, such as those capable of modifying/replicating human voices or producing deepfake videos in near real-time, which are rapidly advancing [66, 7, 241, 173].

Second, a `VIMA` attack is inherently difficult to detect due to the immersive nature of VR systems. Because VR interfaces prioritize full immersion[1], they avoid using service indicators

1. In VR, users are fully immersed in a single, highly engaging 3D environment that surrounds them from all sides,

and security icons typically found on traditional 2D mobile or desktop computing environments. Currently, there is no indicator or reliable means for VR users to understand/verify who or what is controlling their immersive experience (visual or auditory).

In this paper, we demonstrate that the VIMA attack is already viable and effective on popular VR platforms today. In particular, we describe our implementation of a VIMA attack on Meta Quest 2/3/Pro headsets. We show how attackers can alter a variety of user inputs and immersive interactions, including financial transactions in the default browser app (Meta Quest Browser) and live audio conversations in the popular VRChat app for social interactions. We include screenshots and links to the video clips to illustrate these attack instances, highlighting their feasibility and capability to produce harm.

Our VIMA attack design employs a methodology different from previous GUI confusion attacks targeting web browsers and smartphone apps [168, 95, 169, 31, 78]. Prior GUI attacks aim at replacing the foreground app's UI. They rely on either injecting GUI overlays or covertly switching a background app to the foreground – both of which are infeasible in modern VR systems. Instead, our attack constructs a malicious VIMA app to **masquerade as the full VR system**, exploits VR's natural operating flow to seamlessly transition the user into the VIMA app, and traps the user inside this app undetected. We also develop a Unity program to generate the VIMA app customized for each target user under 90 seconds.

We evaluate our attack instances via IRB-approved user studies, covering the whole attack process, from seamlessly transitioning a user into the VIMA app to hijacking their interactions with servers/users. In our primary user study, the VIMA attack successfully deceived all (but one) of 27 participants, including highly experienced users who use VR devices everyday. In post-mortem interviews, after being informed of the attack, half of the participants recalled nothing unusual, while the rest recalled minor glitches but attributed them to standard VR performance issues. Nearly all identified glitches were cosmetic and can be minimized or eliminated by fine-

creating the sensation of being physically situated in the target space. This immersive experience is the key selling point of VR systems.

tuning the VIMA app and content.

Finally, we analyze a range of potential defenses. We find that current VR systems lack some of the security mechanisms deployed on traditional devices. Yet, implementing these mechanisms (or new ones) in VR is more challenging due to its immersive nature. This tradeoff leads us to design a multifaceted defense pipeline.

Overall, our work makes **three key contributions**:

- To the best of our knowledge, we are the first to identify, implement and evaluate interaction manipulation attacks in VR. With VIMA, an attacker's capabilities extend far beyond eavesdropping [163]; they can now manipulate and replace immersive social interactions, one of VR's most compelling features.
- We conduct a comprehensive evaluation of the VIMA attack using three IRB-approved user studies, covering diverse scenarios.
- We propose a multifaceted defense pipeline to increase attack costs and reduce success likelihood.

Ethics. Our user studies were reviewed and approved by our institute's IRB office. We disclosed the VIMA attack and potential defenses to Meta, who positively acknowledged our disclosure and later modified the interface to warn users about potential attacks.

3.2 Background and Related Work

In this section, we summarize existing attacks against VR systems, existing GUI confusion attacks targeting web browsers and mobile devices, and how our work differs from them.

3.2.1 Existing Attacks against VR

Malware. Like all other computing devices [239], VR devices are naturally susceptible to malware. Attackers can introduce malware directly onto a VR headset as malicious apps, or infect other devices connected to the headset. In fact, many VR privacy attacks assume a successful injection of malware to collect user data [127, 137, 217].

Privacy Attacks. To create immersive experiences, VR headsets utilize an array of sensors to continuously monitor user activities, which may lead to privacy attacks. Numerous studies show that records of a user's head and body motions can expose sensitive information, including identity, age, gender, ethnicity, emotions, health conditions, and even the video content they have watched [166, 81, 163, 199, 71, 165, 167]. In parallel, researchers have exploited ways to eavesdrop on specific apps or user input. The man-in-the-room attack [243] infiltrated the Bigscreen VR app to allow unauthorized users to invisibly join and observe their private chat rooms. A range of keystroke inference attacks can recover typed content by either observing avatar hands [262], reverse-engineering an app's network packets [225], or analyzing side-channel leakage patterns [273, 138].

Perceptual Manipulation. These attacks aim to manipulate user orientation or movement in VR by altering their perceived environment. Studies have shown that, if these attacks can be implemented in practice, they can deceive users into changing physical locations, cause collision, or induce motion sickness [49, 41]. A straightforward attack implementation is to develop a malicious app that shifts user's virtual locations when they use the app [231]. Others assume the attacker has strong device privileges to modify screen feed of mixed reality (MR) headsets [49] or system configuration files of VR headsets [41]. Finally, recent works studied UI security in augmented reality (AR) because multiple apps can simultaneously place customized content on the AR device screen [48, 218].

3.2.2 GUI Confusion Attacks

The VIMA attack is related to a broader class of GUI attacks, with analogous attacks in the context of web browsers [95, 56] and mobile devices [75, 32]. We summarize these GUI attacks below. In §3.2.3 we highlight their key differences with the VR context and VIMA.

Browser-based GUI Attacks. The closest analogy to VIMA in web browsers is the *clickjacking* attack [168, 95]. By abusing HTML/CSS features, a clickjacking attack overlays a transparent frame onto a legitimate page, deceiving users into clicking on unintended UI components. Modern web browsers prevent these attacks by implementing frame busters [169] (to detect/extract frame overlays) and additional security measures like Content Security Policy [176].

Mobile GUI Attacks. There are similar GUI overlay attacks on mobile operating systems such as Android. Attackers can either place customized overlays to obscure individual UI components or the entire UI window [169, 31], or exploit the Android developer API to obtain partial control over the system task stack and elevate a background process to replace a foreground app's UI [31]. These attacks can enable clickjacking, allowing attackers to steal a user's login credentials or install apps/permissions without user awareness [78]. They are further facilitated by side-channel attacks that monitor UI states in real-time [46].

Multiple techniques were proposed/deployed to resist these attacks by verifying the authenticity of the GUI. These include the Android Window Integrity (AWI) model that tracks foreground UI states and flags suspicious activities [194], app vetting tools using static analysis [31], and displaying trust information of the foreground app and a secret image on the navigation bar [31]. Today, Android OS prevents overlays from obscuring critical components like the status bar, navigation bar, and confirmation dialogs, while allowing developers to specify UI elements to be protected (so inputs from another overlapping UI window are ignored) [32].

3.2.3 *How* VIMA *Differs from Existing GUI Attacks*

Our proposed VIMA attack is similar in intent as the GUI-based attacks discussed in §3.2.2, but its design is quite distinctive given the constraints of the VR platforms.

First, traditional GUI confusion attacks cannot function in VR. Traditional attacks aim to take control of the foreground app's UI. To do so, an attacker app running in the background needs to either embed customized overlays on the GUI or covertly switch itself to run in the foreground. Both tactics are no longer feasible in modern VR systems. Specifically, since VR exclusively renders and displays a single 3D environment and does not support overlays, a background app cannot place any overlay over the current UI. Similarly, because a VR app running in background cannot render its 3D environment, switching an attacker app into the foreground would cause a very noticeable disruption to the UI.

Thus we take a different approach to implementing VIMA. First, we propose designing the VIMA app to masquerade as the full VR interface rather than the GUI of a specific app. Second, we choose to launch the VIMA app at the end of a benign app session, aligning with the user's expectation of returning to the home environment. This ensures that the UI change appears natural and expected, making the transition both seamless and covert. Finally, once the user enters the VIMA app, they are trapped inside the app unknowingly. Conceptually, VIMA shares a similar intent like the "inescapable" fullscreen attack on Android smartphones, which was hypothesized in [31] but never designed or implemented.

Another key distinction between VR platforms and traditional operation systems is that VIMA-based attacks will be more challenging to secure in VR platforms compared to prior GUI-based attacks. First, VR platforms lack the broad range of mechanisms that authenticate endpoints and information flow available in desktop, mobile and browser-based operating systems. For example, there are no abstractions that authenticate VR connections to VR providers, analogous to SSL connections, CAs and server certificates. Second, VR systems are designed to be fully immersive, a goal that often conflicts with requirements of visible security indicators. Navigating these tradeoffs

to design usable and effective security indicators in VR is an active and open challenge.

3.3 The VIMA Attack

In the context of VR, we define a VIMA attack as:

> *An attack where the attacker controls and manipulates the user's interaction with their VR system, by trapping the user inside a single, malicious VR app that* ***masquerades as the full VR system****.*

A VIMA attack inserts an attack layer between the user and any external entities, behaving as a hidden layer of indirection. While the user thinks they are interacting normally with their home environment, apps and other users, they are interacting with a *single* malicious app, where everything they input, see and hear can be intercepted, relayed, and possibly altered by the attacker. These include both user input (e.g., voice, motion, keystrokes) and app output (e.g. content from app servers, actions of other users). Therefore, the VIMA attack has a significantly broader scope and higher potential for damage than existing VR attacks (§3.2.1). Furthermore, the VIMA attack targets the full VR system rather than the GUI of a specific app, and follows a different design methodology compared to existing GUI confusion attacks (see §3.2.3).

Next, we discuss the threat model considered by our work and provide an overview of the attack design and the key challenges.

3.3.1 Threat Model

The Target. We consider a target user who has a standard, clean VR headset with developer mode activated.

There are three reasons why this is quite reasonable. First, developer mode is commonly supported today by a number of VR platforms (Meta Quest, VIVE Focus and PICO), both to improve usability and to unlock appealing features such as adjusting headset resolution, capturing screen

content and debugging. Developer mode is not perceived as a sensitive configuration like root access, as it neither grants root access to the headset nor permits any root-level operations. In fact, widely adopted enterprise-grade VR management solutions, such as ArborXR[2] [21], ManageXR [142], and Meta Quest for Business [156], all require enabling developer mode to function. Second, there are already reports of millions of VR users today actively enable developer mode for expanded features [211]. Finally, for Android devices, developer mode is a persistent setting that, once activated, remains enabled until manually deactivated.

The Adversary. We assume the adversary does not have physical access or root access to the target's VR headset, but can leverage developer mode to run apps and scripts on the target's VR headset. Later in §3.4.2, we discuss two ways to achieve this on today's Meta Quest headsets and present user study results that validate their practicality. We also assume the adversary has reasonable computing resources to build VR apps, an external server to communicate with the attack app, and their own standard VR headset.

An Alternative Threat Model. We focus on the above threat model as it presents a realistic scenario and reflects the most likely conditions for the attack. Later in §3.8, we describe the attack design and performance under an alternative threat model where the target user persistently disables developer mode on their VR headset.

3.3.2 Overview of the Attack Process

We design the `VIMA` attack by completing a set of sequential tasks. In the following, we discuss the goal of each task and its high-level design. Later in §3.4 we discuss the detailed implementation of each task on Meta Quest headsets.

(1) Build the `VIMA` app: In this task, the goal is to build a `VIMA` app customized for the target, which can mimic the operation of the full VR system. We achieve this by the attacker first collecting headset configuration information and then using it to compile a `VIMA` app for the target.

2. ArborXR is now used by more than 3000 major companies to manage VR devices [4].

Figure 3.2: Screenshot of an example 3D home environment with a menu panel for apps, captured on a Meta Quest Pro headset.

The final VIMA app includes a replicated 3D home environment (see Figure 3.2) and the simulated versions (or proxies) of the apps installed on the target headset. Upon activating the VIMA app, the user is sent to the replicated 3D home environment.

To reduce overhead/latency, we propose to automate the production of the VIMA app: first creating a generic template offline (i.e., a one-time effort), then customizing it to produce individual VIMA apps tailored to different users on the fly. Our attack implementation (§3.4.3) includes a Unity program that generates a customized VIMA app in 90 seconds on a MacBook Pro 2019 laptop.

(2) Activate the VIMA attack: The goal of this task is to lead the user into the VIMA app unknowingly. The most effective method would be to directly launch the app (i.e. entering the replicated home environment) whenever the user puts on the headset. However, this is impractical as it requires modifying the app launcher or bootloader, which requires physical or root access to the headset.

Instead, we achieve this using a "natural transition" taking place during the user's normal VR operation. Leveraging the bootstrapping methods discussed in §3.4.2, the VIMA app is installed on the headset, together with a shell script running in the background. This attacker script detects

when the user signals the system to exit a current app. Normally, this signal would prompt the system to return the user to the home environment. But the script intercepts and destroys this signal, terminates the current app and initiates the `VIMA` app. This silently transitions the user into the replicated home environment and starts the `VIMA` attack.

Our decision to activate `VIMA` during an app exit is deliberate, as an app exit creates a natural (and expected) disruption in the user's immersive experience. As such, the attack can begin seamlessly while avoiding user suspicion.

(3) Hijack user interactions: Once the headset enters the replicated home environment, the interactions between the user and the VR system will go through the `VIMA` app, including the other VR apps the user sees and uses. Thus the attacker can control and modify user interactions with any "simulated" app displayed on the replicated home environment. The level of control over an app depends on how the app is "simulated."

(4) Trap the user in the `VIMA` app: By applying the natural transition method described in task (2), our attack ensures that exiting any app will return the user to the replicated home environment. This traps the user inside the `VIMA` app.

3.3.3 Key Design Challenges

Implementing the above tasks in practice faces two key challenges.

Achieving high accuracy. The first challenge is constructing the `VIMA` app to accurately masquerade as the full VR system, convincing the user that they are interacting with the original VR system. In other words, it involves replicating the 3D home environment and simulating the installed apps on the target headset so accurately that the user does not become suspicious.

Incurring low overhead. The second challenge is how to build the `VIMA` app with a low overhead. In particular, simulating all installed VR apps might seem overwhelming, because replicating the development efforts for each app is impractical.

In the next two sections, we present our detailed attack implementation that addresses both

challenges.

3.4 Attack Implementation on Meta Quest VR

We now present an implementation of the `VIMA` attack on Meta Quest headsets, applicable to all three device versions (Quest 2/3/Pro). In this section, we detail the implementation of the first two tasks in the attack pipeline. Later in §3.5, we describe the third task using two specific apps: Quest Browser (2D) and VRChat (3D).

To provide context, we first describe the workflow and features of Meta Quest headsets (§3.4.1), and how the adversary gets to install and run apps and scripts on the target's Quest headset (§3.4.2).

3.4.1 Preliminaries: Meta Quest VR

Meta Quest headsets run on Horizon OS [154], a modified version of Android OS. After putting on the headset, a user sees their 3D home environment and a menu panel of apps (Figure 3.2). The user clicks on an app icon from the panel to enter the app, and returns to the home environment after exiting the app, typically by pressing the "home" button on the right controller.

As outlined in the threat model, we assume the target has activated developer mode on the headset. Note that developer mode does not grant root access to the headset or allow any root-level operations.

Android Debug Bridge (ADB). Enabling developer mode *automatically* activates Android Debug Bridge (ADB), preparing the headset to accept ADB commands. ADB is a command-line tool for managing Android devices (including Quest headsets), enabling functions such as running shell scripts, installing apps, reading device settings, and transferring data wirelessly. Disabling ADB requires turning off developer mode.

With developer mode enabled, a remote machine (e.g. a peripheral) can command the VR headset by establishing a wireless ADB connection to the headset. In addition, an app that packs

ADB client and server inside its library can also establish ADB access once installed (i.e. by behaving as a wireless ADB connection) [197]. Today, it is increasingly common for apps and peripherals to acquire and leverage wireless ADB access to provide enhanced functionalities on Android devices [179, 128, 204]. Many of these apps (e.g. [128]) have millions of active users.

3.4.2 Attack Bootstrapping: Acquiring ADB Access

Our threat model assumes that the attacker can install/run apps and scripts on the target's VR headset. We now present two methods to achieve this, both by obtaining ADB access to the headset. We also present a user study to evaluate their effectiveness.

Method A: an attacker's machine acquiring wireless ADB access to the target headset. In this method, an attacker laptop gets on the same WiFi network as the target headset and requests wireless ADB access to the headset. Such access requires no credential-based authentication [205] and the user only sees a simple, one-time pop-up[3] requesting access. Once the user accepts the request, the attacker can install apps and run shell scripts in the background without their awareness. They can configure background scripts to run continuously even after the attacker machine leaves the WiFi network (verified by our experiments).

Method A allows the attacker to proactively target a specific target by getting on the same WiFi network.

Method B: an attacker app acquiring ADB access to the target's headset. Alternatively, the attacker compiles an app (i.e., an initial, benign version of the `VIMA` app) with ADB client and server packed inside. The attacker disguises this app as a useful utility app, publishes and promotes it on popular app sites like SideQuest [211]. A user downloads and installs the app on their headset. Upon first launch, the app requests ADB access as it is required for the app to function. Once the user grants this access, the app can now silently execute shell scripts in the

3. All the ADB requests display a "Allow USB debugging?" pop-up and the requesting device's RSA fingerprint. This information is insufficient for users to identify the device type and owner, as shown by [205].

background and establish communication with an attack server. These scripts remain active even after the user closes the app. In addition, via an app update, the malicious version of the VIMA app customized for this user gets downloaded and installed onto the target headset. Compared to Method A, this method is more reactive, but achieves broader attack coverage.

User study on effectiveness. We validated the effectiveness of these two methods using an IRB-approved user study. We recruited 27 participants with different levels of VR experiences, ranging from professional to entry-level. Each participant experienced 6 different attack scenarios, 3 using Method A and 3 using Method B. We recorded their response to the access request (approve/deny) in each scenario, and their explanation for the decision in our post-session interview. For brevity, we list the detailed study setup and results in Appendix §3.11.1, and summarize the key results below.

Key Results. Out of 27 participants, 26 (96%) allowed ADB access in at least 1 scenario, 20 (74%) allowed access in ≥ 3 scenarios, and 8 (30%) allowed access in all 6 scenarios. Between the two methods, Method A has an average approval rate of 46% across all participants/scenarios and Method B has 79%. This difference is because Method B's ADB access request originates from a local app that needs it to function, rather than from an external peripheral. Overall, these results show that attackers can leverage either or both methods to effectively bootstrap VIMA.

A Note on Attack Disclosure. After we disclosed the attack and defenses to Meta, Meta modified the ADB access request pop-up to include a warning of potential attacks (Figure 3.9). Our above user study was done using this updated version, showing that both methods are still effective despite the explicit warning.

3.4.3 Constructing the VIMA *App*

Since the attack proceeds almost identically for both bootstrapping methods, we present the attack implementation assuming Method A, and highlight differences in implementation when applicable.

Leveraging ADB access obtained during bootstrapping, the attacker runs a shell script in the

background to quickly collect headset configuration, and uses them to compile a `VIMA` app customized for the target user. This customized app contains a replicated 3D home environment used by the target user and simulated versions of the apps installed on the target headset. We now describe the detailed technique to construct this `VIMA` app, addressing the challenges of achieving high accuracy and minimizing overhead.

3.4.3.1 Collecting Headset Configuration

The attacker needs three configuration data: the 3D background used by the home environment, the list of installed apps, and the state information. Since VR headsets store 3D backgrounds as APK files, they only need the APK filename used by the current home. This is done via the ADB command `adb logcat`, which returns a log of system activities including the name of the background APK file in use. Since 3D backgrounds typically come from Meta Quest's built-in options or SideQuest Custom Home [212], once identified, the attacker can obtain the actual APK file from their own headset or SideQuest. Similarly, `adb shell cmd package list packages` returns a list of installed apps on the headset. Knowing the app names, the attacker can obtain their button images, from either app APKs on their own headset or online sources like Meta App Store. Finally, `dumpsys` returns the state information, such as recently used apps.

These configuration data can be sent to the attacker server (Method B) or directly viewable on the attacker's laptop (Method A).

3.4.3.2 Replicating the 3D Home Environment

With headset configuration data, one can replicate the target's home environment by replicating immersive content (the 3D background and app menu), monitoring/responding to user inputs, managing app launch/exit, and displaying/altering device settings.

Replicating immersive content is streamlined: unpack the background APK file to retrieve the 3D background, arrange app buttons in a grid, display state information like the recently opened

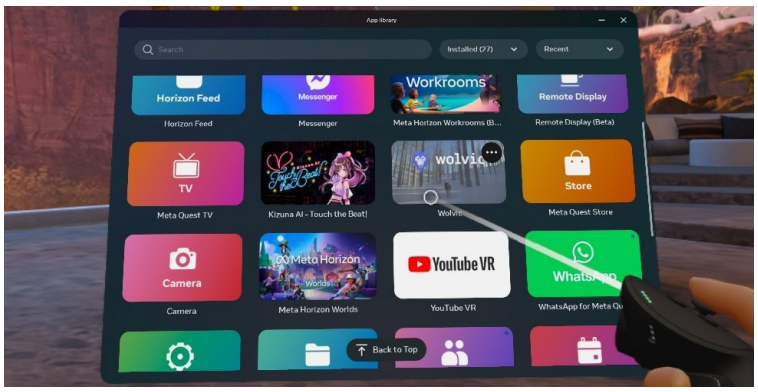

Figure 3.3: The app library window in the Meta Quest Pro VR home. The user is selecting an app called wolvic.

apps, and replicate static elements (cursors, pointers, menu bar) with precision. Recognizing/responding to user input is also streamlined using Meta's Interaction SDK [159] that allows developers to integrate user interaction into their app.

Managing app launch/exit is also streamlined by monitoring user input. When the user clicks on app A's button on the home menu (Figure 3.3), the `VIMA` app will transition into running a simulated version of A. When the user exits an app (by pressing the controller's home button or the virtual exit button), the attack script (running in the background) intercepts the exit signal and transitions the user to the replicated home environment.

Finally, since access to key device elements (WiFi [14], Bluetooth [157] and audio [151]) are automatically granted to all apps, `VIMA` can display/change these settings in the replicated home. Additionally, user information (e.g. friend list) can be accessed via Meta XR Platform SDK [5].

Removing app splash screen. One notable distinction between an app and the native home environment is that an app often displays splash screen at launch time (e.g. all apps created with Unity include one by default). This is not the case for the home environment. When packaging the `VIMA` app, we eliminate the splash screen by using the free Unity Asset Bundle Extractor to adjust app parameters [109].

3.4.3.3 Simulating the Installed Apps

This is a key challenge in our attack design – simulating all installed apps may seem daunting, as replicating the development efforts for each app is impractical. To address this, we show that efficient app simulation is achievable through four distinct methods, varying in complexity and hijacking capability. In practice, full replication of all apps is unnecessary; instead, the attacker applies strong simulations for key target apps and uses simpler methods for others.

In the following, we describe the four simulation methods, with increasing complexity and hijacking capability. An attacker shell script is running in the background to support these simulations and to transition the user back to the `VIMA`'s replicated home whenever the user exits an app (see §3.4.4).

Direct app call (no hijacking). This is for apps that the attacker does not wish to control, but needs to replicate to avoid suspicion. When the user chooses to launch an app, the `VIMA` app will proceed to call the actual app (installed on the headset). An app calling another app is a key Android feature [16] with ready-to-use plugins [24].

Replicating app GUI (stealing credentials). If the attacker only wants to eavesdrop on the target's credentials on an app, they only need to replicate the app's login page. After the target inputs their credentials on the replica, `VIMA` loads the original app to the foreground. Notably, if the original app includes a splash screen during loading, this will lead to an extra splash screen that may raise suspicion. One way to mitigate is to first display an app crashing pop-up and then load the app. Given the common occurrence of Meta Quest apps crashing during loading, this helps minimize suspicion. Finally, upon obtaining the credentials, the replica switches to the dormant mode (i.e. directly loading the original app) until there is a need to record credentials again.

Local simulation via API calls (full hijacking). This aims to realize the full hijacking on the app. Although the attacker generally lacks access to the source code, we design an efficient simulation by replicating the app's GUI and API calls. This is because many VR apps opt to use public APIs [6] to streamline development and ensure cross-platform compatibility [96], by

fetching data from servers to display on the headset.

In this case, the app replica first clones the app's GUI and user interaction, similar to the process of replicating 3D home (§3.4.3.2). It calls the public APIs to communicate with the app server to fetch data, using the target's credential obtained via the replicated GUI. After receiving server data, the replica updates the GUI to display content on the headset. The end result is that the attacker can hijack the target's app session in multiple ways: (1) eavesdrop on communications between the target and the app server; (2) change the target's input to the app; (3) modify app data displayed on the target's headset. Later in §3.5.1 we use this method to simulate and hijack the Meta Quest Browser app.

Over-the-network simulation (full hijacking). We design an over-the-network simulation, where the attacker uses a separate VR headset to run the target app. Now the app replica includes two coordinating components: one on the target's headset and another on the attacker's headset connected to a computer via USB. The app replica on the target's headset implements the app GUI and user interaction (like the above), from which it obtains the target's credentials for the app and forwards them to the attacker's computer using standard network connections. Using the target's credentials, the attacker's headset runs the original app to interact with the app server and relays the relevant content to/from the app replica. As such, the attacker's headset/computer combination acts like a MITM between the target and the app server, allowing the attacker to manipulate the interaction between the two. Later in §3.5.2, we present the detailed simulation of the 3D VRChat app using this method.

3.4.3.4 Automating `VIMA` Customization

We design a Unity program to automate the construction of the `VIMA` app for each target, enabling near real-time app generation.

Offline, the attacker prepares a generic `VIMA` template (for Meta Quest), covering standardized UI components (menus, cursors, user input handling). The attacker selects a set of VR apps

to target, builds their simulated versions (using one of the latter three simulation methods) and packages them into the template. The attacker also gathers app thumbnails and 3D backgrounds from sources like SideQuest [211] and the Meta App Store.

Upon receiving device configuration data, the Unity program creates customized UI components (e.g. app shortcuts in the app menu window) and integrates them into the template. This process is fast, taking under 90 seconds on a MacBook Pro 2019. In our experiments, the `VIMA` app is around 700MB, which is comparable in size to most VR apps and within the official 4GB limit.

3.4.4 Activating the `VIMA` *Attack*

To activate the attack, the attacker first needs to install the malicious `VIMA` app on the target headset and then transitions the user into the `VIMA` app without raising suspicion. The detailed process depends on the method used to obtain ADB access (Method A or B in §3.4.2).

Method A: Through wireless ADB access, the attacker laptop sends the `VIMA` app to the target headset via `adb push` and installs it via `adb install`. The attacker then executes a background shell script (Algorithm 3.10 in Appendix) on the headset, which activates the `VIMA` attack at an optimal moment. This background script continues to run after the attacker leaves the WiFi network.

Method B: Once the initial, benign `VIMA` app is installed and gains ADB access, it executes a background script to collect headset configuration and transmits it to the attacker server. The server prepares a malicious `VIMA` app customized for the target. The benign `VIMA` app on the headset notifies the user to update the app, and when the user agrees, it goes to a predefined link to download/install the corresponding `VIMA` app. In parallel, the benign `VIMA` app launches a background script (Algorithm 3.10) on the headset, which will activate the `VIMA` attack at an optimal point (see below).

The background script to activate the attack. As outlined in §3.3.2, the optimal moment to covertly transition the user into the `VIMA` app is upon them exiting an 3D app, where they expect to return to the home environment, and the attack brings them back to the replicated home environment without raising suspicion. This is achieved by the aforementioned script (Algorithm 3.10) that is running in the background. This script monitors the headset activity using ADB commands `getevent` and `dumpsys`. Upon detecting any signal of an app exit, it proceeds to kill/stop subsequent activities and activates the `VIMA` app.

Trapping the user inside `VIMA`. This script runs even after the attacker disconnects the wireless ADB and the `VIMA` app is closed. As such, it continuously performs the same action whenever it detects an app exit signal, effectively trapping the user inside the `VIMA` app.

3.5 Detailed Hijacking Attacks

To demonstrate the feasibility and impact of interaction hijacking, we simulate two popular VR apps in `VIMA` and use them to perform multiple hijacking attacks. These are Meta Quest Browser [155] and 3D VRChat [245], a social interaction app with more than 20 million monthly users.

3.5.1 Hijacking Meta Quest Browser

As the built-in browser for Quest headsets, Meta Quest Browser offers immersive web browsing experiences. Users can navigate through interactive websites, enjoy streaming videos on a captivating theater-sized screen, and engage with social media platforms seamlessly. Its popularity has motivated web developers to build attractive, novel web content with immersive experiences.

App Simulation. Since the browser app uses public APIs to communicate with web servers (e.g. HTTP requests to access websites), we build its replica using the "local simulation via API calls" method in Unity and package it into our `VIMA` app. Our effort mainly focuses on replicating the browser GUI, displaying website content, and detecting and responding to user inputs.

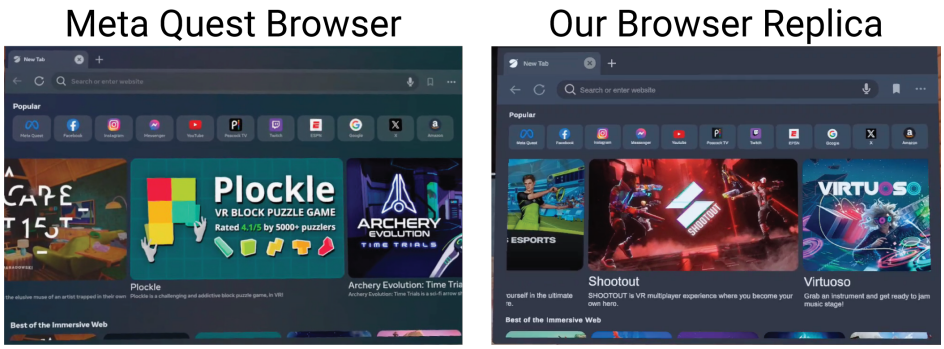

Figure 3.4: A side-by-side comparison of Meta Quest Browser and our replica.

We clone the browser GUI using Unity UI [235], creating a canvas object as the browser window. We configure it to display various UI elements, including texts, images, buttons, toggles, webpages and more. Unity UI also handles registration of user interactions. For instance, each button element has a `OnClick` function to define the effect of clicking it. To display web content, we use Android WebView library [13, 248] (`loadUrl(the_URL)`) to fetch the content of a website at `the_URL`, and display the content at the corresponding location on the canvas. To access user inputs, we use Meta's Interaction SDK [159] to monitor and log user's clicks, drags, and keystrokes on each webpage and their exact locations, and respond to these actions by updating the GUI content and/or sending API calls to web servers to fetch desired contents.

Figure 3.4 provides a side-by-side comparison of the Quest Browser and our replica. Our replica closely mirrors the real version.

Specific Hijacking Attacks. Using this replica, we successfully executed three attack instances: eavesdropping, modifying website content displayed on the headset, and modifying user input to the browser, all in real-time. We describe each below, including screenshots of attack results.

Attack 1: Eavesdropping on sensitive data: The target uses Quest Browser to access sensitive accounts in banking, corporate, and medical websites. The replica can intercept and log private data, including credentials entered by the target. This is because, using Meta Interaction SDK, the replica can accurately track and record cursor movements, keystrokes, button presses, and headset motions. It also has complete knowledge of the website's visual organization. Thus, the attacker

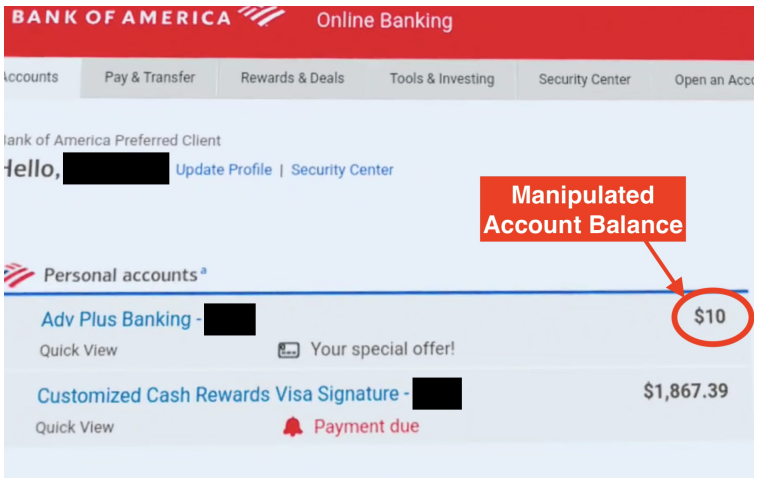

Figure 3.5: In a banking scenario, the Bank of America server sends the correct account balance to the headset. Our attack modifies this balance to display $10 on the headset screen.

can extract user inputs to specific web entries.

Attack 2: Manipulating displayed content (e.g. bank balance): The replica browser can display a modified version of the website content fetched from the server. Consider a banking session. When the target uses the replica to access a bank website, the replica first collects their credentials from its GUI and sends the credentials to the bank server via HTTP requests. After verifying the login credentials, the bank server returns the target's account information to the headset, including the account balances. While all of these network communications are encrypted using SSL handshakes, the content to be displayed on the headset is encrypted using a key provided by the replica during the handshake. Thus the replica can decrypt and obtain the raw content, and can modify them (e.g. using JS code) before displaying them on the headset. Figure 3.5 shows a screenshot of the target's headset display, where we altered their Bank of America account balance to $10 using few lines of JS code (see Appendix 3.11.3).

Attack 3: Manipulating target's input (e.g. payment amount): The attacker can modify user input to the browser replica before using them to generate API calls to web servers, as these API calls typically use plain text and numerical values in the parameter fields.

Figure 3.6 demonstrates a scenario where the target initiates an e-payment through Zelle, a digital payment service of Bank of America. The target requests a $1 transfer by completing the

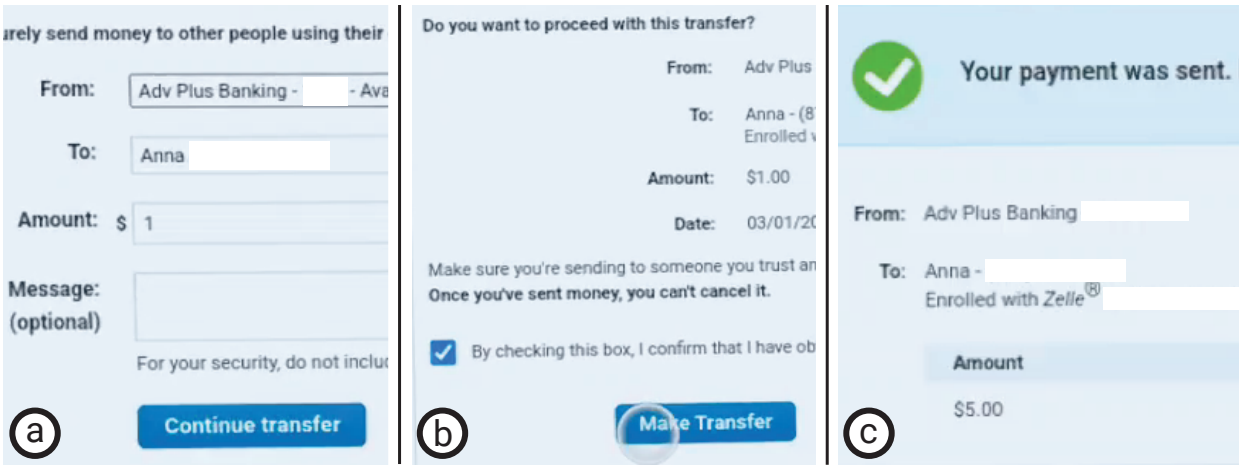

Figure 3.6: In a Zelle transaction, the transfer amount submitted by the target is altered by our attack before reaching the bank server. (a) The target inputs a $1 transaction on the web form. Our attack secretly alters it to $5 before sending it to the server. (b) The target is then taken to a confirmation page to finish the transaction, where our attack sets the amount to $1 on this page to avoid user detection. (c) The actual transaction amount is $5, confirmed by the bank.

web form and clicking "Continue Transfer" (Figure 3.6a). This takes the target to a confirmation page (Figure 3.6b) to finish the transaction by clicking "Make transfer". Using a few lines of JS code (Appendix 3.11.3), we successfully modify the transaction amount to $5 and modify the amount displayed on the confirmation page to be $1. Thus the HTTP request sent to the bank server will request $5 while the user perceives that $1 is requested. Finally, Figure 3.6c shows the legitimate bank confirmation where the actual transaction made is $5. This page should also be modified by the attacker in a practical attack. We leave it this way to demonstrate the success of the attack.

3.5.2 Hijacking VRChat

In §3.1 we described a `VIMA` scenario where Alice and Madison's VR date night session is hijacked by Carl. We now describe an implementation of this attack on Quest devices by simulating the 3D VRChat app [245], where the attacker can listen to and modify live audio conversations between Alice and Madison. Note that while the attacker hijacks the interaction of two users, they only need to launch the `VIMA` app on one (i.e. Alice).

We choose VRChat because it is the leading platform for VR users to interact through personalized avatars, with more than 20 million monthly users [182]. In its immersive 3D environments, participants engage in conversation, gaming, and romantic encounters like date nights [77]. To safeguard player privacy, VRChat uses private APIs for communication between players and servers.

Over-the-Network App Simulation. We replicate VRChat using over-the-network simulation (§3.4.3.3). We setup a laptop (Macbook Pro 2019) and an additional Meta Quest headset connected to the laptop via USB (see Figure 3.7 where the attacker wears the headset). This combo acts as a MITM between the target and the VRChat server. In the following, let Alice be the target of the `VIMA` attack and Madison be the one interacting with Alice in VRChat. Figure 3.8 sketches the attack flow where attacker Carl hosts the laptop/headset combo, in comparison to the benign flow.

To prepare for the attack, the replica on Alice's headset can first operate under the "Replicating app GUI" mode (see §3.4.3.3), i.e. replicating the GUI of VRChat to capture Alice's credentials on VRChat. Using this information, the attacker headset runs a legitimate VRChat app using Alice's credentials, which communicates with the VRChat server to discover any customization (e.g. the list of Alice's favorite VRChat worlds stored in her account). This information is used by the `VIMA` app to fine-tune the VRChat replica.

Next time when Alice starts a VRChat session in `VIMA`'s fake home environment, the attacker headset will "shadow" her activity in the real VRChat app. Specifically, the replica on Alice's headset captures her live speech and motion, livestreaming them to the attacker laptop. As shown by Figure 3.7, the laptop then feeds this data (or a modified version) into the attacker headset as Alice's input to the legitimate app. Simultaneously, the laptop captures the attacker headset's screen display and incoming audio, i.e. Madison's audio and avatar, livestreams them (or a modified version) to the replica on Alice's headset, who replays them to Alice via the GUI.

Overall, this process allows the attacker laptop to control the visual and audio content experienced by Alice and Madison, thus hijacking their VRChat session.

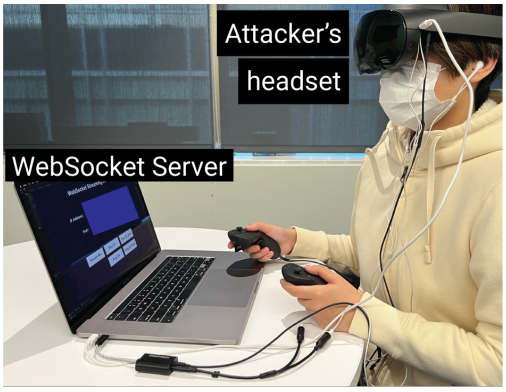

Figure 3.7: A physical setup of the attacker laptop/headset combo. The laptop runs a WebSocket server to receive Alice's live audio, and feeds it into the attacker headset by mounting the earbuds of a wired earphone near the headset microphones. Meanwhile, the audio output of the attacker headset (i.e. Madison's speech) is fed to the laptop via an audio cable and then livestreamed to the VRChat replica on Alice's headset.

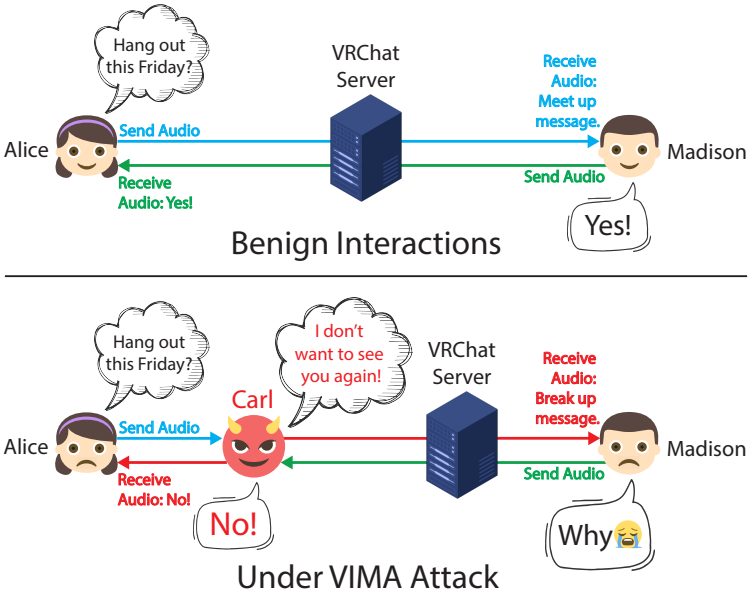

Figure 3.8: The attacker Carl hijacks Alice and Madison's VRChat session. Carl monitors and modifies live audio communications, so Alice and Madison hear different conversations. For example, Carl made Alice hear Madison answering "No" to her question of "Hang out this Friday?" although Madison responded "Yes"; Carl crafted Alice's speech of "I don't want to see you again!" and sent it to Madison to enforce the breakup.

Key Challenges and Our Solutions. The key challenges in this pipeline are the livestreaming performance (delay/jitter) and the laptop's ability to modify visual/audio data in real time. Our current `VIMA` prototype makes the following simplifications.

To implement livestreaming, we set up a streaming server on the laptop using Unity's FMETP STREAM library and configure the replica app on the target's headset to connect with the streaming server via WebSocket. To reduce bandwidth cost, we livestream audio data but not visual frames. In this case, Alice sees Madison's avatar whose body motion is predefined in the replica app. In the setup where the replica app communicates with the attacker streaming server via a 5GHz WiFi network, the livestream delay is within [0.4,0.6] seconds in both directions. This setup was used in our user study (§3.7).

The task of modifying visual or audio data in real time is more challenging. While our attack design can naturally utilize editing tools, the current prototype adopts a simpler, immediate solution for audio editing: using pre-recorded speech data of Alice/Madison. These data can be recorded from prior VRChat sessions or generated offline by high-quality voice synthesis models. We discuss the practical benefit of adopting real-time editing in §3.7.2 and leave full integration with such tools to future work.

Figure 3.11 in Appendix shows screenshots of what Alice and Madison see in their headsets. We evaluate this attack setup using a user study (§3.7), demonstrating its effectiveness at deceiving users.

3.6 User Study on Efficacy of VIMA Attacks

We perform a user study to evaluate the efficacy of the VIMA attack in deceiving target users. In particular, we aim to understand two key research questions:

- RQ1: What are the target user's reactions and responses during the process of a VIMA attack?
- RQ2: What are their self-reported reactions and reflections after knowing the occurrence of the VIMA attack?

Ethics. To evaluate our VIMA attack in a everyday setting, we withhold information about the attack from the participants at the beginning of the study. We inform them that the study aims

to investigate user experience in interacting with VR apps. This deception study is evaluated and approved by our institutional IRB. No personal data of the participants is retained post-study.

3.6.1 Study Setup

Participants. We recruited 27 participants from our institute (P1-27), including 5 females, 21 males, and 1 participant who chose not to disclose. The participants range in age from 20 to 30 years old and have varying degrees of familiarity with VR devices. We prioritized finding participants with experience in VR. Among them, 3 participants are experts who use VR devices on a daily basis (P1-3); 6 participants are professional users, regularly using VR devices on a weekly basis (P4-9); 10 participants are knowledgeable users who have had multiple experiences in using VR devices (P10-19); 8 participants are "entry-level" with no prior experience (P20-27). Our study lasts roughly 30 minutes per participant. Each participant is compensated with $20. Details on participant demographics are listed in the Appendix (Table 3.4).

Procedure. The study is held inside a large room, involving one participant at a time. We provide each participant with a Meta Quest Pro headset to immerse themselves in VR. Upon putting on the headset, they find themselves in a legitimate VR home environment. We guide them through headset controls and interactions, and allow them to navigate freely until feeling comfortable with the system.

We then invite each participant to explore VR apps and rate their experiences. This task is divided into two parts. In Part I, we ask the participant to access the app library[4] from the VR home and choose one or two apps to explore. After approximately 10 minutes, we ask the participant to exit the current app, which brings them back to the VR home. The participant then opens the Meta Quest Browser app to rate their experiences on a Google form. Here the form also asks for their institutional ID. To protect the participant, the ID is immediately discarded after the current study.

4. The library features 14 popular apps, including Horizon Workrooms, Horizon Worlds, Beat Saber, YouTube VR, and Meta Quest Browser, covering a broad category of office productivity, social networking, gaming, entertainment, to web browsing.

Next in Part II, we ask the participant to repeat the Part I process by exploring two additional apps, exiting the app, then activating the Quest Browser app to fill in the Google form again.

What the participant does not know is that, during Part I, we use WiFi ADB access to secretly install the `VIMA` app on their headset and run the script in the background to monitor the headset activity. Thus when the participant exits the app in Part II, the attack script activates the `VIMA` app and transitions them to the replicated VR home (§3.4.4). Therefore, the Quest Browser app they open in Part II is a simulated version inside the `VIMA` app, allowing us to record their institutional ID.

After the above VR session, we debrief the participant, disclosing details about the `VIMA` attack and its occurrence within the session. We then interview the participant, asking them to elaborate on their awarenesses and observations about the attack, and reflections on their behaviors in the VR session (details in Appendix 3.11.5). After obtaining participant consent, we record their vocal responses to the interview. After analyzing all responses, we discard all the audio recordings to protect participants' identity and privacy.

Analysis Procedure. We followed best practices for analyzing open-ended questions [201]. Two interviewers first independently studied notes and recordings to categorize responses into general themes regarding the interviewees' level of suspicion, level of hesitation, and reasons why. They then met to consolidate their code books into a single one while resolving any discrepancy. The final coding category represents sentiments expressed by > 2 participants.

3.6.2 Results

We first present the key observations.

- All 27 participants proceeded with the session without any observable suspicion or hesitation, except for one VR expert (P3). Prior to our debriefing, P3 voiced suspicion about Part II, stating that "I would likely investigate the matter if it weren't for the user study."

- After the debriefing, 14[5] out of the 27 participants recalled discrepancies between Part I and II. Interestingly, all these 14 participants (except P3) attributed the discrepancies they observed to standard VR performance variations or glitches, which did not concern them at all. For the remaining 13 participants, the attack came as a complete surprise since they did not notice any discrepancy.

- Nearly all of the reported discrepancies were cosmetic (e.g., different cursor size, missing status of recent apps) and can be effectively reduced or eliminated by fine-tuning the `VIMA` app and improving the precision of context replication. We provide details on the reported discrepancies in Appendix 3.11.5.

Next, we delve into the reasons for the lack of suspicion, based on the participants' reflections on their behaviors during the VR session.

Reasons for No Suspicion. Our analysis yielded two key explanations for both highly experienced and less experienced VR users.

- **Accustomed to fluctuating VR operations and glitches.** P1, P2, P4, and P5 who are highly experienced VR users stated that they are accustomed to fluctuations and glitches in VR operations, and tend to perceive any unusual patterns as standard system variations or glitches. Additionally, P1 elaborated that the frequent updates to VR systems contribute to the difficulty in recognizing (ab)normal operations – "*Even if I notice some differences [in* `VIMA`*], I wouldn't think they are fake. I will be like [the company] probably did some [system] updates*."

- **Harmless discrepancies.** Some less experienced users noticed differences between the `VIMA` phase (Part II) and the normal phase (Part I). However, they do not perceive them as suspicious or harmful in any way. For example, P13 noticed that the status of recent apps is missing but stated that "*I don't think it was suspicious or anything*." P14 recalled

5. These 14 participants include 2 expert, 5 professional, 5 knowledgeable, and 2 entry-level VR users.

that the controller cursor is a bit different, but considered the difference benign because "*the interaction still works like when I want to click something, it still works*."

Reasons for Disclosing Personal Information. We are also curious about why all the participants disclosed their personal information (institutional ID) in Part II. There are two noteworthy reasons.

- **Trusted settings.** Many have developed trust in reputable enterprises and believe that products offered by these companies are safe to use. In our study, some participants expressed confidence in the safety of using Google Forms on the Meta Quest Browser, considering that both products are supplied by well-established enterprises, e.g., "*I mean the browser seems legitimate. It was again the Meta browser and [...] in the particular Google form so it did not seem like it was an attempt to get my data*." (**P24**).

- **Repetitive behaviors.** People tend to autopilot with momentums and habits when it comes to repetitive activities, e.g., we usually key-in the password to unlock our smartphones without hesitation. In our study, each participant responded to a Google form twice, first on a legitimate browser and then on a malicious replica. Several participants mentioned that they thought the second time was the same as before so they just repeated what they did without hesitation. "*I feel like I was going through the same route as before [...] so I do not assume [...] that would steal my identity*." (**P14**).

Takeaways. Our user study results demonstrate the efficacy and potency of the `VIMA` attack, which successfully deceived 26 out of 27 participants. Even highly experienced users, who use VR devices daily/weekly, were susceptible.

These results also show challenges for users to detect and resist `VIMA` attacks. The inherent volatility and glitches in today's VR systems make it challenging to notice minor discrepancies or flag them as suspicious. Also, trust placed on popular apps developed by reputable enterprises,

along with habits formed from previous app usage, encourages users to swiftly repeat usual operations on these apps, without questioning their authenticity.

3.7 Additional User Study: Hijacking VRChat

We performed another user study on the scenario of hijacking a VRChat session. Different from the user study in §3.6 where each participant is the target of the `VIMA` attack (i.e. Alice), the participant in this new study is the VR counterpart (i.e. Madison) who interacts with the `VIMA` target. We aim to understand a new research question:

- RQ3: How do VR counterparts react to the target's interactions that are altered by the `VIMA` attack?

Ethics. Following the same reason in §3.6, we withhold information about the attack from the participants at the beginning of the study, and inform them that the study aims to investigate their VRChat experiences. This deception study is approved by our institutional IRB and no personal data of the participants is retained post-study.

3.7.1 Study Setup

Participants. We recruited 3 groups of participants from our institution (7 females, 5 males; age: 23-31). Details on participant demographics are listed in Appendix (Table 3.6). Each group has 4 participants. Group 1 (P1-4) are experienced VR users (one expert and three professional VR users). Group 2 (P5-8) are less experienced VR users (two knowledgeable and two entry-level VR users). Group 3 (P9-12) are less experienced VR users, but interact with our interviewer on a daily basis and thus are familiar with their voice, tone, and speaking habits. We explain why we recruited these 3 participant groups below.

Procedure. Each participant uses a Meta Quest Pro headset to join a legitimate virtual world in VRChat (a virtual beachside cabin), represented by an avatar in this virtual world. We guide

the participant through the controls and interactions of VRChat and the avatar, allowing them to explore freely in this VRChat world. Then, we leave the room and the participant is in the room alone.

After approximately 5 minutes, the interviewer (one researcher in a different room) joins the same VRChat world using their own avatar. The participant now sees the interviewer's avatar and hears their voice. The interviewer then asks the participant to answer 4 questions about their experiences/thoughts on VR systems (e.g., how often do you use VR headsets?). We design these questions to be simple so that the participant is able to pay more attention to other elements in this VRChat interaction, for instance, potential changes in the audio quality caused by the `VIMA` attack.

What the participant does not know is that the interviewer in their VRChat world is under the `VIMA` attack and thus hijacked by an attacker[6] throughout the whole interaction. Following the over-the-network implementation discussed in §3.5.2, the interviewer is using a VRChat replica built by the attacker, and the attacker now relays or modifies the live audio sent to the participant's headset. Specifically, for the first two questions (i.e. Part I), the attacker relays the live audio of the interviewer; for the next two questions (i.e. Part II), the attacker replaces the live audio with a pre-recording of the interviewer asking the same questions.

After the above session, we conduct an in-person interview. We ask each participant to elaborate on their experience in the session, and any observation that stood out to them (details in Appendix 3.11.6). After obtaining participant consent, we record their vocal responses to the interview. We follow the same analysis procedure discussed in §3.6 and discard all audio recordings post-analysis.

In this study, we intentionally recruited 3 groups of participants with varying VR experiences and varying familiarities with the interviewer's voice. This helps assess how VR counterparts with different knowledge of the target would react to the altered interactions.

6. The attacker is the other researcher in a room away from the participant. The attacker's laptop and headset are connected to our institute's 5GHz WiFi network, like the headsets of the interviewer and the participant.

3.7.2 Results

Of the 12 participants, 10 expressed no suspicion. The 2 (P11, P12) who voiced suspicion were from Group 3, both attributing their concern to Part II, although their reasons for suspicion differed. For P11, it was the lack of response – P11 requested clarification on one question, but the attacker had no suitable prerecorded response thus did not reply. Consequently, P11 was confused: "In Part II, I felt like I was having a one-sided conversation." For P12, it is because one of the (prerecorded) responses felt too generic and indifferent. One explanation is that individuals who are familiar with each other expect more engaging conversations and interactions.

P9 and P10, also from Group 3, noted that Part II had slightly better audio quality than Part I but did not consider this discrepancy suspicious. This is because the live audio in Part I naturally includes background noises and echoes, while our pre-recordings were made in a quieter environment. This can be addressed by adding real-world artifacts into the pre-recordings to better represent live conditions.

Takeaways. Our study shows that in VR social interactions (using VRChat), a `VIMA` attacker can effectively deceive the target's VR counterparts using pre-recorded speech audios of the target.

Meanwhile, attackers need to invest effort to forge/alter speech audios in real-time in order to reliably deceive users who are familiar with the target and who expect more engaging conversations and interactions. Given the rapid advancement of generative AI, it is likely that attackers now have, or will soon obtain, tools capable of producing highly realistic audio that mimics any user's voice. Notably, RESEMBLE.AI just launched a paid "voice-changer" tool that transforms one voice (e.g. the attacker's live speech) into another (e.g. the target's speech) with a 100ms delay [7]. This brings `VIMA` attackers one step closer to engaging with the target's VR counterparts on their behalf, i.e., hijacking their VR interactions.

3.8 Alternative Threat Model & Attack Design

So far, our attack design assumes that the target's headset has activated developer mode. We now consider a weaker threat model that disables developer mode and discuss attack design and implications.

A weaker threat model (developer mode off). With developer mode disabled, the target's VR headset can only install non-system apps from the official app store (designated by the headset manufacturer). These apps cannot run shell scripts.

The attacker has no access to the headset (physical, root, or ADB). We assume the attacker is able to build a generic `VIMA` app as a seemingly benign app with hidden `VIMA` components. The attacker successfully publishes it on the official app store, and the target user downloads and installs the `VIMA` app on the headset.

Attack design. Unable to run shell scripts, the `VIMA` app cannot observe headset activity beyond its own. We propose to overcome this by embedding a "trigger" inside the `VIMA` app that activates the hidden attack components when the user runs the app. In particular, we propose to include a custom-made "exit" button for the `VIMA` app. When a user clicks this button to exit the app, they are transitioned into the simulated home environment hidden inside the `VIMA` app. This activates the `VIMA` attack like the original design in §3.4.4.

Challenges and implications. Due to the inability to run shell scripts, the attack faces two key challenges/limitations.

The first is gathering device configuration to customize a `VIMA` app for the target user. On Meta Quest headsets, this can be (partially) addressed since apps can gather the list of installed packages (apps and background files) using a standard API call [15] and send them to the attack server as standard communication. The server uses the Unity program (§3.4.3.4) to produce an updated `VIMA` app, and asks for an app update before the next use. One remaining caveat is that when multiple background files are present on the target headset, the attacker cannot determine

which one is in use, but needs to choose one, e.g., based on popularity. A wrong choice could lead to user suspicion, although some might attribute it to standard system glitches and proceed as usual.

The second challenge is keeping the user trapped inside the VIMA app. Without the script, the VIMA app cannot intercept exit calls. Thus pressing the controller's home button gets the user out of the VIMA app, until they run the app again. This will limit the attack frequency/duration, but will not prevent it completely.

3.9 Potential Defenses

Having demonstrated the practicality and effectiveness of the VIMA attack, we now discuss potential defenses, their efficacy and potential limitations. As a maturing technology, current VR systems lack many of the security mechanisms found on desktop and mobile systems. We categorize potential defenses into three key groups: *attack prevention*, *detection*, and *hardware-based mitigation*.

After examining a broad set of potential strategies, we propose a multifaceted defense pipeline to both increase the cost of launching the attack and reduce the likelihood of its success. Table 3.1 lists the specific elements in this pipeline along with their attack counterpart. For brevity, we describe these elements below and include a detailed discussion of other strategies in Appendix 3.11.7.

3.9.1 Attack Prevention

We first present defense measures aimed at preventing the installation, activation or execution of the VIMA attack.

Preventing Installation. The obvious choice is ***disallowing developer mode***, forcing the attacker to publish the VIMA app on the official app store and wait for users to install it (§3.8). Today, Apple

Vision Pro headsets disallow developer mode while Android (Meta Quest) devices support it.

Given that developer mode is an essential feature widely adopted by Quest users, fully disabling it on Android-based headsets could be challenging. Thus it is important to ***mandate informative tutorials*** on the security risks associated with developer mode, and to ***minimize the need for enabling developer mode*** by migrating important use cases to more controlled environments. While these approaches are more realistic, their protection against the `VIMA` attack relies on user choices, making them less reliable.

Even with developer mode disabled, the attacker can still activate `VIMA` by publishing a seemingly benign `VIMA` app on official app stores (§3.8). It is crucial that ***app stores conduct rigorous testing on apps and their updates*** before publishing them. In particular, thorough manual checks can help identify apps that exhibit abnormal behaviors on exiting or those that display home environments.

Preventing Activation. Once the `VIMA` app is downloaded onto the headset, a viable defense is to prevent it from getting activated. One approach is ***enforcing app certificates***, where the headset requires digital certificates to authenticate and verify the identity of an app before allowing it to run. In this case, the attacker must either obtain a certificate or exploit system flaws to bypass validation [171], significantly increasing the effort and costs required.

Disrupting Attack Execution. Now assuming the `VIMA` app is running, we can make it hard and/or costly to operate. Similar to how secure task management could mitigate task hijacking [196], one method is to ***prevent non-system apps from calling other apps***, so the `VIMA` app must replicate apps using full simulation (§3.4.3.3) or deploy complex shell scripts to handle app calls. This raises the attack cost (and latency). However, disabling app-to-app transitions faces practical drawbacks. For development, apps often need to initiate other apps and fork processes, especially for multi-scene apps. Disabling this feature increases development costs and reduces app usability.

A practical alternative is to ***validate authenticity of app calls***, similar to the client authentica-

Attack Elements	Defense Elements
Inject the VIMA app (and the shell script)	- Disallow developer mode (DM) or Minimize the need for DM - Rigorous app tests by app stores
Activate the VIMA app	- Enforce app certificates - Validate authenticity of app calls
Run the VIMA app	- Validate authenticity of app calls - Anomaly-based attack detection - Regular headset restarts

Table 3.1: Elements of our proposed defense pipeline against VIMA attacks and their attack counterpart

tion process to resist network MITM attacks [79, 209]. VR headsets can include a source validation process to authenticate app communications/transitions, forcing the attacker to either use full app replicas or break the authentication process. Both increase the attack overhead. This validation should also be applied to app calls initiated by shell scripts, making it harder to trap a user inside the VIMA app. Yet this authentication-based defense is not entirely foolproof [79, 209].

3.9.2 Attack Detection

Once the attack is in progress, the practical defense is to quickly detect its presence and shut it down. The key to attack detection is identifying anomalies produced by the attack process. Detection can be automated by performing static analysis on the app [31] or by monitoring app and device's runtime behaviors [195, 31], or it can be performed manually by the user themselves.

Generally speaking, given the inherent complexity of usage behaviors and fluctuating VR performance, achieving reliable attack detection is highly challenging. Furthermore, VR's immersive and captivating experiences can completely grab and hold the user's attention, leaving them with reduced alertness and awareness of any subtle indicators of VIMA attacks.

Below, we illustrate two most promising directions and the challenges they face. Appendix 3.11.7 present additional potential directions and their limitations.

Detecting short-lived home environment. When the user presses the controller's "home" button to exit an app, the system automatically initiates an activity call to bring the legitimate home

environment to the foreground. Under a VIMA attack, the shell script (§3.4.4) terminates this activity and pushes the VIMA app to the foreground. This cause the legitimate home to be short-lived. This system behavior, if happened frequently, indicates the presence of the VIMA attack and identifies the attack app.

The attacker can choose to not always intercept the signal of "home" button, e.g. periodically put the attack script to sleep. This can disrupt the pattern for detection, but also prevents trapping the user in the VIMA app indefinitely. Alternatively, the attacker can design app replica to encourage the user to press a virtual "exit" button (controlled by the attacker) to exit an app.

Comparing system and user's perceived app traces. The OS tracks headset activities to generate an app usage trace and report the sequence to the user. If this sequence differs from the user's recollection, they will be alerted. For example, the system logs the VIMA app, while the user recalls opening VRChat. This defense depends on user awareness and decision-making, making them less reliable. Also how/when to present this data to VR users poses a challenge, as it inevitably disrupts the immersive experience.

3.9.3 Hardware-based Mitigation

Hardware defenses provide an offline layer of security independent of potentially compromised software. We discuss several hardware strategies in the Appendix (§3.11.7). The most viable strategy is ***regular headset restarts***. A restart will kill the process running the VIMA app and the shell script making the attack dormant until it is activated again. Although it does not remove the VIMA app and the script, it mitigates potential harm by reducing the active time of the attack. A downside is significant interruptions to the user experience, since users prefer to resume their progress from their last VR session. Along this line, an extreme strategy is ***regular hard reset*** that removes all apps.

3.10 Conclusion

We introduced VIMA, a new, powerful VR interaction manipulation attack. We described an implementation on Meta Quest headsets, which can eavesdrop and modify what VR users see/hear and what they communicate with servers/users. The result is a wide array of personalized misinformation attacks, from misrepresenting bank balances, changing value of financial transactions, to modifying audio conversations of two interacting VR users without their awareness. Results of our user studies validated the potency of these attacks in real world settings.

Our work emphasizes the need to strengthen security measures in VR systems to significantly decrease the frequency and impact of VIMA attacks and similar threats. But the clock is ticking. New generations of VR hardware will boost computing power and enable more powerful interaction-based attacks, e.g. an attacker replacing a VR user with seamless, real-time injection of a generative AI version of their avatar. VR platforms and developers need to act now to improve security and educate VR users about potential risks.

3.11 Appendix

3.11.1 User Study on Acquiring ADB Access

In this user study, we examine how users react to ADB access requests produced by our two bootstrapping methods (Method A and B in §3.4.2). Specifically, we aim to understand two key research questions:

- RQ1: What are the user's decisions (approve/deny) upon receiving ADB requests initialized by Method A and B?
- RQ2: What are their self-reported factors affecting their decisions?

Ethics. This study is approved by our institutional IRB. No personal data of the participants is retained post-study.

Type of VR Users	Participant	Age	Gender
Professional	P1	26	F
	P2	31	M
	P3	27	M
Knowledgeable	P4	21	M
	P5	27	M
	P6	26	M
	P7	28	M
	P8	26	F
	P9	21	M
	P10	24	M
	P11	37	P
	P12	26	M
	P13	28	M
	P14	26	F
	P15	27	M
	P16	25	F
Entry-level	P17	30	M
	P18	20	M
	P19	30	F
	P20	22	M
	P21	26	F
	P22	23	F
	P23	52	F
	P24	22	F
	P25	27	F
	P26	21	F
	P27	27	F

Table 3.2: Demographic information of the participants in our user study described in both §3.4.2 and §3.11.1. F: female. M: male. P: prefer not to disclose.

Participants. We recruited 27 participants from our institution (P1-27), including 12 females, 14 males, and 1 participant who chose not to disclose. The participants' ages range from 20 to 52. They have varying degrees of familiarity with VR devices. Specifically, 3 participants are professional VR users who use VR devices on a weekly basis (P1-3); 12 participants are knowledgeable VR users who have had multiple experiences in using VR devices (P4-16); the remaining 12 participants are entry-level VR users with no prior VR experience. We list detailed demographics in Table 3.2.

Procedure. The study is held inside a conference room, and involves 1 participant and 1 researcher at a time. At the beginning of the study, the researcher informs the participant that the goal is to understand their interactions with VR apps. They are encouraged to act freely and make their own decisions in each scenario. We then provide the participant with a Meta Quest Pro headset to experience 6 attack scenarios in VR. Prior to each scenario, the researcher provides a consistent description of the scenario, including its location and tasks, while deliberately omitting

	Location	Urgency	Engagement	Task	ADB Access	Approval Rate
1	Train station	–	Medium	Watch 360° scenic videos	Method A	41%
2	Airport	–	High	Play an interactive game	Method A	44%
3	University	–	High	Design a 3D backpack with a classmate	Method A	52%
4	Airport	Medium	–	Adjust framerate to save headset's battery	Method B	74%
5	Office	High	–	Use an app recommended by a coworker for a meeting	Method B	74%
6	Home	Low	–	Disable unnecessary notifications	Method B	89%

Table 3.3: The 6 scenarios evaluated in the user study, involving common public and private locations and representative tasks like gaming and setting headset configurations. We also vary the levels of task urgency and engagement to understand their potential impacts. The scenarios were consistently described by a researcher to all participants, ensuring uniformity across the explanations. ("–" means not applicable.)

any details related to security or ADB access to minimize bias.

The 6 scenarios involve common locations ranging from public to private environments and representative tasks ranging from gaming to setting headset configurations (Table 3.3). The first 3 scenarios are evaluating Method A and the rest are evaluating Method B:

- **Scenario 1-3:** Based on Method A, the attacker requests ADB access over the same WiFi network at random moment (here the researcher secretly using a random timer generator), assuming zero knowledge of the target's (here the participant's) current states in VR. Intuitively, the more engaged the participant is in their VR experiences, the less likely they are to scrutinize the access request. Therefore, we design the tasks with different levels of engagement.
- **Scenario 4-6:** Based on Method B, the attacker requests ADB access via a benign version of the `VIMA` app, which disguises itself as a utility app. The app informs the target that it needs ADB access to properly function. Naturally, the more the participant wants the utility, the more likely they approve the access. Therefore, we design the tasks requiring utility apps with different levels of urgency.

Upon receiving the request, a system pop-up appears on the participant's VR screen (Figure 3.9). On the pop-up, the participant can choose one of the three options: "Cancel", "Allow", and

"Always allow from this computer." The researcher quietly watches the participant's VR screen content casting to a laptop and notes down the decision.

After completing all six scenarios, a second researcher joins the room. Together, the two researchers interview the participant, verbally describing the scenarios again, reminding the participant of their decisions, and asking them to elaborate on the reasoning behind their decisions. The researchers document the participant's responses and, with the participant's consent, record the audio of the interview. To protect participants' identity and privacy, all audio recordings are deleted after the responses have been analyzed.

Analysis Procedure. When analyzing the study results, we followed best practices for analyzing open-ended questions (using the same analysis procedure described in §3.6).

Results. We hereby present our key results.

- Only 1 (P12) out of 27 participants refused to allow ADB access in all six scenarios. During the interview, P12 voiced their cautiousness about security and worried about harming the VR device if they granted the ADB access.
- Eight out of 27 participants allowed ADB access in all 6 scenarios.
- Twenty out of 27 participants allowed $\geq$ half of the scenarios. These include P1-2 who are professional VR users.
- Method A has an average approval rate of 46% across all participants/scenarios and Method B has 79%. This difference is because Method B's ADB access request originates from an installed app that needs it to function, rather than from an external peripheral.

We report additional takeaways from the study.

- The level of engagement and urgency has marginal effects on users' decisions. Instead, they are more likely to accept the request when trusted locations are involved. For instance, P11, P16, and P21 rejected Scenario 1 and 2 but granted access to Scenario 3 because they trusted the university environment; Scenario 6 has the highest approval rate but the lowest urgency

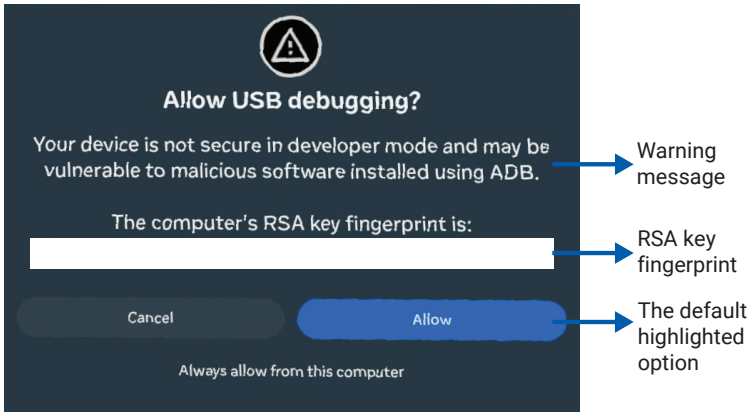

Figure 3.9: Upon receiving an ADB request, this "Allow USB debugging?" pop-up will appear on the Meta Quest headset's screen. Originally, the pop-up only displays the RSA key fingerprint and no warning message about the developer mode and ADB. After we disclosed the attack to Meta, they added the warning message to the pop-up. Our user study in §3.11.1 was done using this updated version, showing that the `VIMA` attack is still effective despite the explicit warning.

level. Specifically, seven participants said they felt safe in a home environment during the interview.

- Users are more likely to accept the request if it aligns with their expectations and could affect their current tasks. As shown in Table 3.3, many participants (including P1-3) granted access in Scenario 4-6, where the app explicitly informs the participant to allow the access request for it to function properly. Based on the interviews, almost all these participants approved the access because they wanted the apps to work for their tasks.

3.11.2 Additional Materials for §3.4.4

Due to security concerns, we do not disclose the complete shell script used by the `VIMA` attack. In Algorithm 3.10, we only show the code snippet of a key function: detecting exit calls and starting the VIMA app.

Algorithm 3.10: A key function of the `VIMA` shell script

```
-----------------------------------------------------
# Listen to input events
getevent -l | awk '

  # If home button press (i.e. an exit call)
  /EV_KEY       KEY_FORWARD          UP/ {

    # Get and kill the current activity
    system("am kill `dumpsys activity |
            grep top-activity |
            grep -o 'com.*/' |
            tr -d '/' |
            tr '\n' ',' |
            cut -d',' -f1`")

    # Start the VIMA app
    system("am start -n VIMA_app")
  }
'
-----------------------------------------------------
```

3.11.3 Additional Materials for §3.5.1

The bank balance alteration is done by executing the following JS code in the replica browser via [248]:

```
document.getElementsByClassName(
    'balanceValue TL_NPI_L1'
)[0].innerHTML = '$10';
```

The following JS code modifies the transaction amount to $5.

```
document.getElementById('btnModalSave')
.addEventListener('mouseover', () => {
  document.getElementById('txtAmount')
  .value = '5';
});
```

And the attacker modifies the amount displayed on the confirmation page via the JS code below:

```
document.getElementById('transfer-amount')
```

```
.innerHTML = '$1.00';
```

3.11.4 Additional Materials for §3.5.2

Figure 3.11 show the screenshots of what Alice and Madison see in their VR headsets during a VRChat session hijacked by the VIMA attacker Carl.

Figure 3.11: The left figure is Alice's view in a VRChat replica crafted by Carl. The right is Madison's view in the legitimate VRChat app, where Alice is impersonated by Carl.

3.11.5 Additional Materials for §3.6

Table 3.4 lists demographic information of participants (P1-P27) recruited for our user study in §3.6.

Table 3.5 reports the mean and standard deviation of the VR home loading time (in seconds), which vary across trials and applications. The results show that the VIMA attack results in an average increase of only 1.5 seconds.

Observed Discrepancies

- P2, P9, P10, and P26 recalled that a system pop-up was missing in Part II. In Part I (no attack), when pressing the "home" button to exit an app, a pop-up will appear and ask the user to confirm the action of exiting the app.

Types of VR Users	Participant	Age	Gender
Expert	P1	30	M
	P2	26	M
	P3	30	M
Professional	P4	27	M
	P5	25	F
	P6	20	M
	P7	25	M
	P8	26	M
	P9	28	M
Knowledgeable	P10	21	M
	P11	25	M
	P12	25	M
	P13	20	F
	P14	25	F
	P15	20	P
	P16	20	M
	P17	27	M
	P18	21	M
	P19	27	F
Entry-level	P20	25	M
	P21	23	M
	P22	25	M
	P23	26	M
	P24	23	M
	P25	20	F
	P26	26	M
	P27	21	M

Table 3.4: Demographic information of the participants in our user study in §3.6. F: female. M: male. P: prefer not to disclose.

	Exiting Application			
	Beat Saber	Meta Horizons	Bigscreen	Rec Room
No attack	8.10 ± 0.68	7.55 ± 0.74	8.30 ± 0.64	8.10 ± 0.68
VIMA	9.41 ± 0.64	8.98 ± 0.73	9.62 ± 0.68	9.69 ± 0.72

Table 3.5: Measured loading time (seconds) of the VR home environment after exiting an app (mean$\pm$std) under "no attack" and "VIMA attack" scenarios, across 10 trials per app. The extra delay ($\approx$ 1.5 seconds) observed under the VIMA attack is spent on loading the app assets and scenes.

- P3, P7, and P13 recalled that the status of recent apps is not displayed in the VR home in Part II[7]. In particular, P3 stated that "*[The fact that] it didn't memorize the previous states that I have [really bothers me].*"
- P9 and P14 reported that the controllers are a bit different. P14 noted that "*I think the [cursor] beam [of the controllers] is shorter [compared to the legitimate one].*"
- P4, P14, P18, and P21 reported that few UI components are less responsive in Part II.

7. Indeed, the simulated VR home used to run this user study did not display the status of recent apps because, at the time, we did not perceive it as necessary or significant.

- P5, P9, and P12 noticed that the loading time of the "VR home" during VIMA is slightly longer.

The first four types of discrepancies can be effectively addressed by improving the replicas of VR elements with greater precision. For instance, the system pop-up is essentially a 2D app, which the attacker can easily replicate. The attacker can access the state information (i.e., recent app history) via an ADB command: `dumpsys` during bootstrapping.

The **loading time discrepancy** is caused by the difference in how VR loads the home environment versus initializing an app. Initiating a 3D app requires decompressing assets and loading the 3D scene, which may take extra time. To gain deeper insights, we measure loading times in both normal and VIMA scenarios, as the duration from the user's initiation of exiting an app to the appearance of the home environment. Here we vary the app being closed and perform 10 trials per app, while ensuring a consistent 100% battery charge throughout the measurement. Table 3.5 in Appendix reports the mean and standard deviation of the loading time (in seconds), which vary across trials and applications. The VIMA attack results in an average increase of only 1.5 seconds. Considering the inherent variability in VR performance over time, such small deviations may not trigger user suspicion. The feedback we gathered from user responses also confirmed this hypothesis.

We note that this time discrepancy only occurs when activating the VIMA attack, because loading the VIMA app takes longer than loading the original home environment. After that, the VIMA VR home loading time is nearly identical to the no attack scenario (since the app assets and scenes are already loaded).

Interview Questions in §3.6

1. Did you observe any suspicious occurrence? If any, why they made you suspicious?

2. Did you observe any unusual occurrence? If any, why they did not make you suspicious?

3. Did you hesitate when disclosing your personal information? Please elaborate.

3.11.6 Additional Materials for §3.7

Interview Questions in §3.7

1. How do you feel about your experience in Part I? The experience includes your interactions with the interviewer, visuals and audios in the VR environment.

2. How do you feel about your experience in Part II?

3. Please name the differences between Part I and II.

Table 3.6 lists demographic information of participants (P1-P12) recruited for our user study in §3.7.

User Group	Participant	Age	Gender
Group 1	P1	28	M
	P2	31	M
	P3	26	F
	P4	26	M
Group 2	P5	28	M
	P6	28	F
	P7	26	F
	P8	26	F
Group 3	P9	26	F
	P10	23	M
	P11	28	F
	P12	24	F

Table 3.6: Demographic information of the participants in our user study in §3.7. F: female. M: male.

3.11.7 Additional Defense Strategies

In our search of defenses against the `VIMA` attacks, we have explored a wide range of potential strategies and formulated a multifaceted defense pipeline, as described in §3.9. Below, we discuss additional potential defense strategies that were not included in the proposed pipeline.

Defenses via Prevention. Here we list additional defense measures aimed at preventing the installation, activation or execution of the `VIMA` attacks.

- **Prevent installation.** We present an additional strategy to prevent VIMA from installing by ***adding secure authentication to networking ports.*** Intuitively, requiring stronger authentication on networking ports would limit the installation of VIMA attacks through unauthorized remote connections. However, this protection is ineffective against insider attacks (e.g. in enterprise settings) where the attacker has already obtained the authorization. Furthermore, as discussed in §3.4.2, publishing the VIMA app with ADB embedded can still obtain ADB access (since the device requesting it is the headset itself). In both cases, secure authentication fails to provide protection.

- **Prevent activation.** Another mechanism under this category is to ***enforce the kiosk mode***. In the enterprise setting, turning on the kiosk mode can restrict the set of apps that the user can interact with. But it significantly limits the flexibility and increases operational costs. We also note two additional caveats: 1) some enterprises allow employees to use personal headsets for work, and this defense cannot be implemented; 2) the machine/server that manages the use of the kiosk mode is not immune to the attack: if compromised, the entire system would fall under control by the attacker.

- **Prevent VIMA app from calling other apps.** In addition to the techniques discussed in §3.9.1, we also consider ***enforcing uni-processing for 3D VR apps***, which is already in place on Meta Quest VR headsets. Due to performance limitations, Meta Quest devices only support the running of a single 3D environment at a time and immediately stop the previous 3D app when launching a new one. Therefore, if the VIMA app chooses to directly call a 3D app (§3.4.3.3), the VIMA app itself will stop. However, this has little/limited impact on the attack since the shell script (§3.4.4) will reactivate the VIMA app when the current 3D app exits.

- **Prevent user access to OS shell.** The VIMA attack runs shell scripts to detect when the user exits an app and then activates the VIMA app, and to collect configuration information of

the headset to replicate the home environment at a high precision. Disabling user access to the OS shell blocks an attacker from executing these scripts, which would reduce the effectiveness and stealthiness of the VIMA attacks. However, doing so also disables legitimate headset users from communicating with the OS, e.g. the user can no longer run customized processes, kill processes, or change system settings. Thus, disabling OS shell access could be highly challenging to enforce in practice. Finally, this defense cannot stop the alternative version of VIMA attacks (§3.8).

Defenses via Attack Detection. Attack detection can be done automatically by monitoring and profiling app/device behaviors or manually by the VR user. Here we discuss in detail three additional directions and their limitations.

- **Control flow monitoring.** When app calling is supported for non-system apps, it enables the VIMA app to call other apps to implement low-cost app replication. A detection mechanism can exploit app behaviors, as the VIMA app potentially raises suspicion with more frequent and diverse app calling than legitimate apps. However, apps have complex and user-driven flows, which poses challenges to benchmarking a benign control flow measurement. Also, as VR integrates more into workflows, app calling between legitimate apps may become more common, making the distinction harder to detect. Hence, detection mechanisms relying on control flow may generate many false alarms.

- **Performance profiling.** Performance statistics like delay and power consumption may increase when the VIMA app is running. Measuring these metrics may provide information that aids detection. The system could monitor parameters like CPU/GPU usage, memory access patterns, system calls, API calls, etc., to establish a baseline, and alert if performance deviates significantly. However, with the unpredictable nature of user behavior, these metrics are inherently noisy and the detection mechanism is likely inaccurate. Moreover, the attacker's ability to adapt can introduce further challenges, as the VIMA app could be carefully crafted

to mimic the performance profile of benign apps, making detection more intractable.

- **Educating users.** Users might notice subtle anomalies but dismiss them as bugs or glitches (see our user study in §3.6). `VIMA` may be detected if users are aware of this form of attacks and are alert to minor changes in appearance or experience. Furthermore, users can introduce frequent customization of VR home features, e.g., background, to make it harder for the attacker to replicate their home environment at a high precision.

  On the other hand, user self-detection of cyber attacks tends to be challenging and unreliable in general. Specifically for VR, with users accustomed to imperfect VR systems, this defense is unlikely to be effective. Here we also note that the immersive and captivating experiences provided by VR systems could "overwhelm" the users and diminish their alertness and awareness of any subtle indicators of the `VIMA` attacks. And new generations of VR systems (through either hardware or software advances) will only amplify the effect of overwhelmingness even further.

Defenses via Hardware. In addition to regular restarts, another defense technique on the hardware side is ***regular hard resets***. Hard-resetting the headset wipes the device and hence completely removes the `VIMA` app and the shell script. The adversary would need to inject the `VIMA` app and the script again to attack the user. We note that wiping the device significantly disrupts user experience, hence this defense is quite extreme.

CHAPTER 4

DEFENDING AGAINST ADVANCED BIOMETRIC SPOOFING

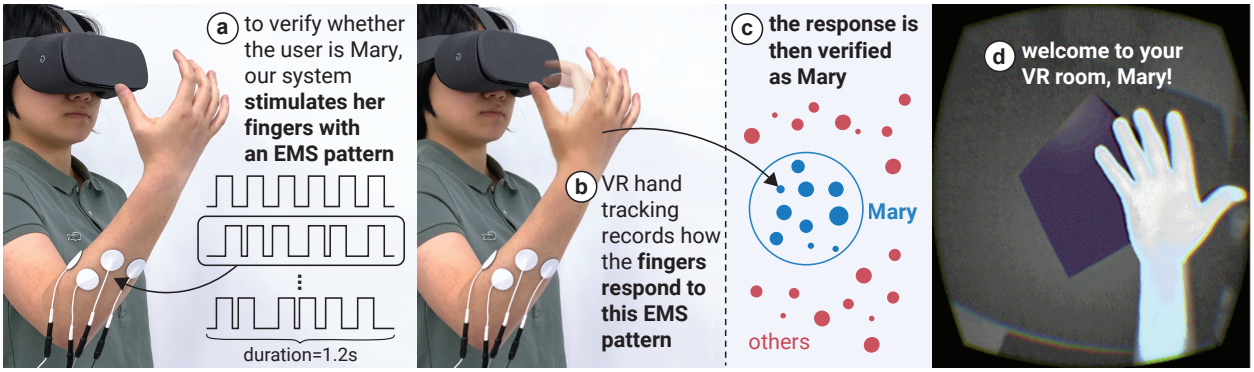

Figure 4.1: We propose a novel modality for authentication: electrical muscle stimulation (EMS). To explore it, we created an interactive system that (a) stimulates the user's forearm muscles with electrical impulses (i.e., using one of 68M possible EMS challenges); (b) measures the user's involuntary finger movements, which are unique because everybody's physiology is different; (c) verifies this response using an authentication model, and immediately eliminates this challenge, making our system secure against data breaches and replay attacks as it never reuses the same challenge. We demonstrate it here using the example of (d) authenticating a VR user without passwords or PINs.

4.1 Introduction

Biometric authentication is a technique that identifies an individual by their unique biological characteristics, such as their iris [252], fingerprints [140], or even one's voice [39]. To identify their users, these interactive systems compare a previously stored biometric key to incoming, typically real-time, biometric data of the user wishing to authenticate. Compared to traditional password or PIN based systems, biometric authentication offers significantly better usability as it does not require users to memorize passwords or PINs. As such, biometric authentication is getting widely adopted, replacing passwords in many contexts [222].

However, the key feature of biometric authentication is typically also its key flaw: once the biometric data is compromised (e.g., stolen in database breaches or recorded by an external attacker), there is nothing the user can do to securely re-use their own data. For example, if someone steals a

user's fingerprints, this user can never trust a fingerprint-based interactive system. Unfortunately, these threats are not theoretical and many biometric systems have been breached. For instance, the biggest known biometric data breach involved a database of 27.8M records, including fingerprints and faces [98].

To tackle this shortcoming, researchers turned to interactive systems that feature a *challenge-response* as a form of active biometric authentication. One example is Velody [118], which challenges a user by vibrating her palm and measuring the user's unique vibration-response. The advantage of these systems is that, if the stored challenge-response pairs are breached, the system can quickly recover by simply asking the user to submit responses to a new set of challenges. As such, researchers seek to find more modalities that afford challenge-response biometric authentication.

In this paper, we propose and explore a novel modality for active biometrics: electrical muscle stimulation (EMS). To understand and evaluate the potential of EMS as a biometrics system for interactive applications, we engineered a prototype that performs user authentication via EMS. Our system, which we call ElectricAuth, stimulates the wearer's forearm muscles with an EMS-based challenge, i.e., a 1.2s long sequence of electrical impulses on four of the user's muscles. Then, it measures the user's involuntary movements that result from this EMS challenge. In Figure 4.1, we illustrate our system with the example of authenticating a user in VR. Here, ElectricAuth uses the VR headset's hand tracking to observe the response of the user's muscles to the EMS-challenge as their individual finger muscles are actuated.

ElectricAuth makes three key contributions in the design of EMS-based biometric authentication.

First, ElectricAuth authenticates users by leveraging what is typically seen as the biggest disadvantage of EMS: *intersubject variability*, i.e., the same electrical stimulation results in different movements in different users because everybody's physiology is different [110, 58, 63, 51, 160]. This unique response to EMS across users is well-known and well-documented in the early HCI works that pioneered the use of EMS in interactive devices, for instance: "(..) stimulation level

differed between users and was clearly dependent on the muscle and fat level and thickness of the arm" (from Kruijff et al. [114]) and, similarly, "(...) levels according to individual variations" (from PossessedHand [229]). In fact, researchers in the field of muscle-biomechanics and physiology demonstrated how this uniqueness arises from multiple factors, such as differences in muscle contractility [84], muscle elasticity [233], muscle viscosity [57], the limb's mass and shape [170], skin conductance [121], bioimpedance [55, 203] and even nerve conduction [8]. All these differences add up to create individual responses to the same stimulus, which our system uses as the key feature to authenticate a user.

Second, ElectricAuth generates a very large pool of challenges by exploring an underutilized property of EMS: muscles respond differently depending on their current state of contraction, which can be altered by varying the timing between two impulses. Using four muscles, six impulses and seven time gaps, ElectricAuth encodes one of 68M possible challenges in 1.2s. As such, ElectricAuth is robust against data breaches and replay attacks because it never reuses the same challenge twice in authentications – ElectricAuth rejects replay of recorded responses to any previously used challenges, and can quickly recover from leak/breach of either authentication model or stored challenge-response pairs by asking the user to register responses to a new set of challenges (like registering new one-time passwords).

Finally, we evaluated our prototype of ElectricAuth by means of four different evaluations, each shining light on a different facet of our research question: (1) in our user studies, we found that ElectricAuth offers accurate user verification and resists three common biometric attacks: impersonation, replay and synthesis attacks; (2) in our exploratory longitudinal study, we found that ElectricAuth's pre-trained authentication model performed stably over 21 days against various muscle conditions (fatigue, humidity, etc.) that were absent from the training data; (3) in our technical evaluation we showed that ElectricAuth, after receiving a response, can verify the user in 3ms on laptop's CPU and 35ms on a small embedded device; we also confirmed the use of depth camera as an alternative motion tracking modality (since our prototype uses IMUs); and, (4) we

generated synthetic impersonator responses to test ElectricAuth's robustness against impersonation attacks at scales larger than our user studies.

4.2 Related Work

The work presented in this paper builds on the fields of wearables, electrical muscle stimulation, and biometrics.

4.2.1 Electrical muscle stimulation

Electrical muscle stimulation (EMS) is a technique from medical rehabilitation [224] that induces involuntary movements by delivering electrical impulses to the user's muscles. This is typically achieved by non-invasive methods such as attaching pairs of electrodes to the user's skin (e.g., on top of the muscles that control finger movement, located in the forearm). Electrode pairs are typically driven using safe and medical compliant muscle stimulators [111].

The range of motion of an induced muscle contraction depends on several key factors. Even in the very first interactive use of EMS in HCI, by Kruijff et al. [114] in 2006, the potential causes of EMS' intersubject-variability were discussed: "(..) stimulation level differed between users and was clearly dependent on the muscle and fat level and thickness of the arm (...)". Similarly, in PossessedHand [229], Tamaki et al. also found "(...) stimulation levels according to individual variations". In fact, researchers in the fields of muscle-biomechanics and physiology have been investigating precisely which factors drive a muscle's unique response to electrical impulses, including: the location of the electrodes [229, 192]; the electrical waveform characteristics, such as frequency and amplitude of the impulses [229, 192, 114]; the target muscle's contractility [84], i.e., the ability of muscle fibers to shorten; muscle elasticity [89, 233], i.e., the ability of the elastic tissue present in the muscle fibers to return to its original length when a tensile force is removed; muscle viscosity [57], i.e., the internal bio-lubrication of the muscle inhibits the muscle from reacting too quickly to protect against stretch injuries; the limb's mass and shape [58, 63, 170, 51, 43];

skin conductance affects non-constant current EMS devices [121, 114], bioimpedance [55, 203]; and, even nerve conduction [8, 223], i.e., the speed of nerve signal transmission. However, it is not possible to precisely determine how much each factor weighs in the final variability, as these are tied together in complex non-linear ways, and this is still an open research question in muscle physiology. More importantly, all the aforementioned factors are relevant to our proposed technique since these vary-across users. Typically, a combination of these explains the *intersubject variability* seen in EMS-based interactive systems, which is why researchers report long periods of calibration [229, 227, 130, 133] and even specifically mention differences across users [114, 229].

Recently, researchers started to engineer interactive devices based on EMS. These tend to fall into two broad categories: (1) haptic devices that increase immersion/realism of virtual environments, and (2) interactive devices that facilitate information access via proprioception. As far as interactive devices that increase immersion, EMS has been used to render forces in mobile devices [129], virtual reality [130, 133] or augmented reality [134, 72]. As a means of general information access, EMS has been especially used for haptic training (e.g., learning a musical instrument [229], operating a tool the user is not familiar with [132]) or eyes-free communication (e.g., communicating walking directions via leg stimulation [227], communicating a state of a variable via wrist movements [131]).

Unlike these interactive systems that use EMS as a form of force feedback or as an information channel, we explore EMS in a new direction: leveraging user's unique muscular responses to EMS as a form of active biometric authentication.

4.2.2 Biometric authentication

Biometric authentication verifies an individual by their unique biological characteristics. To verify a user's identity, a biometric authentication system compares a previously stored biometric key from a particular user to incoming, typically real-time, biometric data of the user wishing to authenticate. Compared to traditional password or PIN based methods, biometric authentication

offers significantly better usability by not requiring the user to memorize passwords or PINs.

Existing biometric systems can be categorized into two types: passive and active biometrics.

Passive biometrics. Passive biometrics rely on physiological characteristics that naturally occur in users, which can be either static or dynamic. Static data, e.g., fingerprints [90], handprints [83], facial and eye features [140, 252, 175, 9], is often used for authentication. Biometrics based on dynamic data recognize patterns that vary over time, e.g., heartbeats [99], gait [228], mouse movements [105], keystrokes [230], speech features [3], body movements [180, 164], pulse-response [191] and bioimpedance [91, 203]. Compared to static data, these display greater complexity and are harder to model.

Passive biometrics are vulnerable to data thefts and replay attacks as reported by numerous incidents and studies [172, 254, 259, 253, 34, 108, 64]. This is because the identity (also known as "key") associated with each user is physically "hard-coded" and then used *repeatedly* for all authentications. Thus after a key has been compromised (e.g., stolen from a database), an adversary can bypass authentication until the key is replaced. Finally, there is a small number of available biological traits per user that act as suitable keys, e.g., once all ten fingerprints are compromised, this user can never again rely on fingerprint authentication.

Active biometrics via challenge-response. Active biometrics leverage a user's physiological response to a given stimulus (also known as "challenge") injected by the interactive device. The assumption is that each user's response to a given challenge is unique. Thus, each *challenge-response* is effectively a biometric password. Examples of challenge-response biometrics include leveraging: the palm's response to vibrations [118], reflexive eye behaviors in response to visual stimuli [219], or even EEG responses [124]. These systems authenticate implicitly so the user does not need to consciously follow the challenge, e.g., the palm vibrates and the user is authenticated [118].

Compared to passive biometrics, active systems are more robust against data thefts and replay attacks. This is because each user can potentially generate many challenges, each triggering a

different response. The system uses a new challenge in each authentication session, preventing attackers from using previously observed responses to breach it.

Lastly, while many challenge-response authentication systems leverage the user's movement (e.g., gaze [190] or wrist shakes [177]), these require explicit action from the user. Unlike these, our novel exploration of EMS-based authentication provides the advantages of movement-based challenge-response while automatically delivering the challenge and eliciting the user's *involuntary* response.

4.3 Implementation

To help readers replicate our design, we provide the necessary technical details. Furthermore, to accelerate replication, we provide source code and training scripts[1]. Here, we describe in detail the prototype we implemented for our user studies, which is based on sensing the user's movements using inertial measurement units (IMUs). However, this is just one possible configuration for our concept. As depicted in Figure 4.1, other tracking systems, such as optical tracking [249, 153], are likely feasible alternatives.

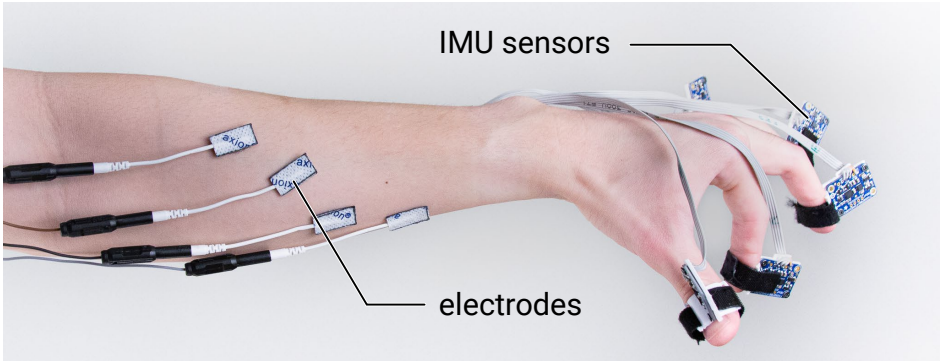

Figure 4.2: IMU-based version of our EMS authentication system, which we used for our user studies.

1. `http://sandlab.cs.uchicago.edu/electricauth`

4.3.1 System Overview

ElectricAuth consists of three components: (1) a medically-compliant **EMS device** that delivers EMS challenges to the user, (2) a **motion sensor** that captures the actuated limb's movements, such as IMUs or depth cameras, and (3) a **trained machine learning model** that classifies the user's movements and performs authentication. Figure 4.2 depicts one concrete implementation of our system using EMS and IMUs attached to a user's forearm, which we used for our user studies.

1. EMS hardware.

EMS stimulator: For delivering EMS impulses we use the Hasomed Rehastim, a medical compliant device with eight individually controllable channels. This device has often been used in interactive systems based on EMS [135, 133, 134]. To control the EMS stimulation, our software sends serial commands via USB using the Hasomed's Science Protocol [85]. These impulses have a latency of <1ms.

Customized EMS sleeve: As with any device based on EMS, we start by calibrating the electrode placement for each user at her registration session. Our calibration aims at targeting four muscles on the user's forearm that actuate finger and wrist rotation. At the anterior forearm we stimulate two muscle groups: (1) primarily the *flexor carpi radialis* and partially the *flexor digitorum profundus*; and, (2) the *flexor pollicus longus*. At the posterior forearm we stimulate two muscle groups: (1) primarily the *extensor digitorum* and partially the *extensor digiti minimi, extensor pollicis brevis & longus*; and, (2) the *extensor indicis*. As is typical with EMS-based systems, these electrode positions are adjusted for each user during the registration session to ensure comfort. Because each user has a different muscular anatomy and body shape, the resulting electrode locations are different across different users.

After calibration, the resulting electrode layout for a particular user is fixed by making an EMS-electrode sleeve (fabric with electrodes stitched to it) that this user wears any time they use ElectricAuth. Moreover, the sleeve becomes part of each user's own challenge definition, i.e., an attacker trying to impersonate a particular user will require obtaining or copying the user's sleeve,

which we later validate in our studies by actually providing the impersonators with the EMS sleeves of the legitimate users.

EMS parameters: Our EMS stimuli on all electrode locations are the same: single-shot square-impulses with an intensity of $10mA$ and a pulse-width of $200\mu s$. We chose this configuration for two reasons. First, we configured EMS impulses to generate small and subtle finger movements rather than large conspicuous movements typical of most existing EMS research, because this enables more practical authentication scenarios. While these smaller movements are harder to recognize, our results suggest that our authentication model can accurately track these (see Section 4.7). Second, we opted to make all impulses uniform to shine light in the fact that intersubject variability in EMS arises from factors external to EMS waveform characteristics.

Our EMS challenges are constructed by sequencing these standardized pulses to one of the four channels the user's forearm is connected to. For instance, one can construct a challenge with a sequence of six impulses, each followed by a resting period. We detail the engineering of our pulse sequences in Section 4.3.2.

2. Motion sensing.

We utilized a set of five 9-DOF inertial measurement units, attached to the fingers via a 3D printed ring (NXP Precision 9DoF, comprised of the FXOS8700 3-Axis accelerometer and magnetometer and the FXAS21002 3-axis gyroscope). These sample the fingers' acceleration and rotation at 50Hz (post-sample interpolated to 100Hz) with a precision of ±4g at 14-bit for acceleration and ±250°/s at 16-bit for rotations; note that we do not use the magnetometer. These IMUs are sampled by a ATSAMD21G18 ARM Cortex M0 48 MHz processor, via a TCA9548A I2C Multiplexer. Finally, our sensing board relays the IMU data via serial over USB to our software.

While attaching IMUs to each finger has been shown to be a reliable way to capture hand pose [94, 60], we believe many alternative tracking systems are possible to realize EMS-based authentication, such as depth cameras [215, 249], RGB cameras [40, 213], and others [107]. We provide a short evaluation that confirmed the use of depth cameras as an alternative tracking system

in Section 4.9.

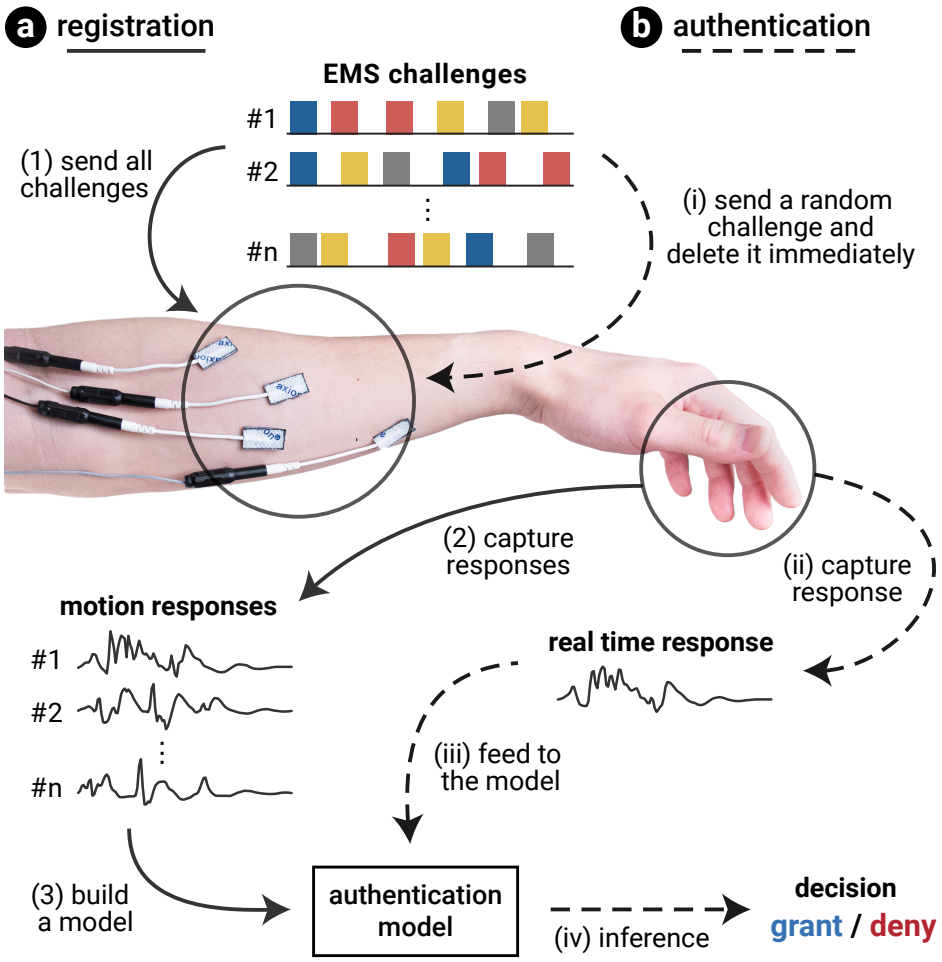

Figure 4.3: Interactive pipeline for the registration (registering a new user) and authentication phase (interactive use in runtime). User response can be caputured using a motion capturing device, e.g., IMUs and cameras (not shown). In this system, the EMS device and electrodes are wearable; the motion capturing device is either wearable or placed near the user; while the authentication model can be remote.

3. Authentication software and pipeline.

The software component of ElectricAuth, written in Python, handles all the interactions between EMS device, motion sensing, model training and real-time authentication. The pipeline of ElectricAuth, which is depicted in Figure 4.3, is comprised of two phases: (a) **registration** and (b) **authentication**.

In the **registration** phase, marked by solid lines in Figure 4.3, registering a new user (after

calibration) is as follows: (1) a set of n EMS challenges are sent one at a time; (2) the user's movements in response to each challenge are recorded; (3) these responses are used to train a machine learning-based authentication model for this user. The number of challenge-response recorded per user is the primary factor that dictates the total time the system needs for registering a single user (we detail this in Section 4.4).

In the **authentication** phase, marked by dashed lines in Figure 4.3, verifying a user's identity in run-time is as follows: (i) one random EMS challenge belong to the claimed identity is chosen, deleted immediately from the database, and sent to the user via EMS; (ii) the user's movements in response to the challenge are recorded; (iii) the motion responses are fed into the trained authentication model of the claimed identity; (iv) the system determines whether this user is legitimate (i.e., being the claimed identity) or not.

4.3.2 Engineering EMS-based Challenges

As our system is the first that explores EMS for authentication, we dedicated a significant part of our exploration in understanding how to increase the challenge pool using EMS; a large challenge pool is what makes a challenge-response based authentication system robust against data breaches and replay attacks. Naively, one can stimulate the user's muscles with individually configurable pulses; however, this (1) requires more calibration time and (2) does not reveal the mechanisms that explain these individual responses. Therefore, we kept purposely all EMS impulses uniform for all users of our system; this grants us more confidence in interpreting the unique responses as originating from the physiological differences between users. Yet, this introduces a challenge when it comes to diversifying the challenge pool.

One straightforward solution (adopted by many existing works on challenge-response biometrics [118, 219, 124]) is to sequence stimuli but separate them by a fixed time gap. If we were to adopt this as well, the maximum number of EMS challenges would be S^L, where S is the number of unique stimuli in the system and L is the number of stimuli in each challenge. For example,

a sequence of six EMS impulses over four possible EMS channels, with a fixed rest period between each impulse, results in $4^6 = 4,096$ challenges. We were interested in whether we could dramatically surpass this approach.

To significantly increase our challenge pool, we explored a rather unused property of human muscles that causes them to respond differently to EMS depending on their current state of contraction. We call this *temporal dependency*.

Temporal dependency. We empirically found, in our preliminary pilots, that a subject's response to an EMS stimulus is affected by the previous stimulus in the same challenge, and the impact depends on the time gap between them (represented as τ).

Figure 4.4 shows two example traces of a finger's acceleration when we stimulate the user's muscles with a sequence of two stimuli (A and B) but vary the time gap between A and B (i.e., $\tau = 0.1s$ and $\tau = 0.17s$). The measured acceleration displays different characteristics when we vary τ. The strongest candidate for a physiological explanation is that muscle contractility and elasticity vary with muscle length [232, 82], and the response to a stimulus depends on the muscle lengths at the time of stimulation. Thus, depending on the gap between A and B, the subject's unique contractility [84] and elasticity [89, 233] will lead to different responses.

The use of temporal dependency affords a large EMS challenge pool by varying the time gaps between consecutive stimuli. Assuming they all produce distinguishable responses, the number of unique challenges (of length L) is upper bounded by $S^L \cdot T^{L-1}$, where T is the number of distinct time gaps. For our ElectricAuth prototype, we utilize $S = 4$ EMS channels and $T = 7$ different time gaps ($\tau = \frac{1}{30}$s, $\frac{2}{30}$s, ..., $\frac{7}{30}$s), which in early pilots we found to lead to sufficiently different movement outcomes. The maximum number of unique challenges is 112 (L=2), $87,808$ (L=4) or $68,841,472$ (68M) (L=6), compared to 16 (L=2), 256 (L=4) or $4,096$ (L=6) when we do not vary the time gap.

Further increasing the challenge pool. Encoding longer challenges is another way to expand the challenge pool. With $S = 4$ stimulus locations and $T = 7$ time gaps, sending $L = 8$

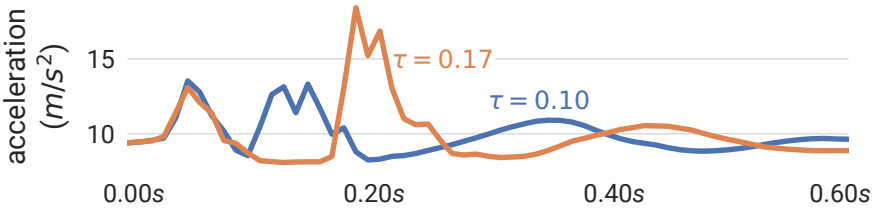

Figure 4.4: An example of how a response changes when the time gap between two EMS stimuli varies: we vary the time gap from 0.1s (blue curve) to 0.17s (orange curve).

pulses (<2s) increases the pool size to $53,971,714,048$ ($4^8 \times 7^7$). Also it is possible to add more electrodes or customize EMS impulses to further diversify the pool.

Checking for uniqueness. Ideally, every challenge-response authentication in the pool is unique. However, in practice this might not be the case given the granularity and sensitivity of motion sensors. To enforce uniqueness, ElectricAuth can apply a verification step during user registration. Specifically, after generating new challenges for a user at the registration phase, it collects the corresponding responses and checks the similarity across these responses and previously registered responses (e.g., computing the mean square error (MSE) between raw responses). If a new challenge is identified as a previously registered challenge, this new challenge is removed.

4.4 User Authentication Model

We now present the design of ElectricAuth's user authentication model. ElectricAuth requires a trained authentication model per legitimate user, which is used to verify whether a test subject is indeed that user. To do so, the model takes as input the response to a given challenge designed for the legitimate user, and outputs whether the test subject is legitimate. Our authentication model was designed with two objectives in mind: (1) minimize the amount of samples collected from the user (i.e., reducing registration overhead) and (2) resist common attacks (e.g., impersonation and replay attacks) and data breaches.

4.4.1 Overview

Initially, we explored implementing our model using specific features of the user's IMU data in response to particular EMS challenges (so called feature-engineering). However, we quickly realized a major downside of this approach: as the response data we capture in real-time from the IMUs is complex (thirty concurrent data streams: 5×3 axes of acceleration and 5×3 axes of rotation), simple feature extraction might not capture the full expressivity of the data. Therefore, after experimenting with this approach, we turned to neural network based models.

We implemented a robust authentication model that, for each registered user, integrates two deep neural network (DNN) models to resist both impersonation and replay attacks. Specifically, authentication starts with **(1) an unsupervised anomaly detector**, which verifies whether a response was produced by the user the model belongs to (i.e., the legitimate user); this step prevents *impersonation attacks*, in which a different user attempts to gain identity of the legitimate user. If a response passes the anomaly detector, it then enters **(2) a challenge classifier**, which detects and rejects *replay attacks* by verifying whether the response is the reaction to the challenge used in the current authentication session.

Both models are trained using only the challenge-response pairs of this legitimate user collected during registration. When the user (re)registers a new set of challenges, we retrain both models from scratch using the new data. This also enables ElectricAuth to recover from data and model breaches.

4.4.2 Detailed Model Design

1. Verifying user via unsupervised anomaly detection.

We implement user verification as unsupervised anomaly detection [36], where the detection model is trained on *only* the legitimate user's responses collected during registration. At run time, the model verifies whether an input response was likely originated by the legitimate user. This anomaly-based detector leverages the fact that responses from other users will display characteristics dif-

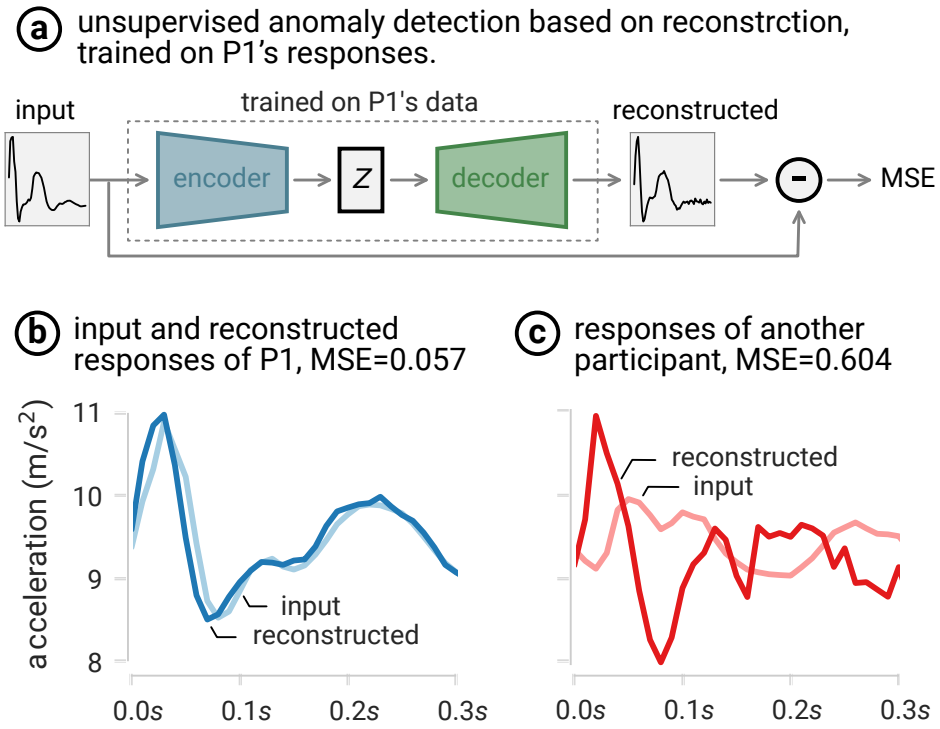

Figure 4.5: Authentication starts with an anomaly detection, which verifies if a response came from the legitimate user that the model belongs to (P1 in this example). (a) The anomaly score is the MSE of the input and model-reconstructed responses. We illustrate how our anomaly detector correctly: (b) identifies P1 (legitimate user) with a low MSE and (c) rejects P2 (impersonator) with a high MSE.

ferent from those of the legitimate user. Thus the model is designed to produce normal output when the input response comes from its legitimate user, but abnormal output when the input comes from any other user. This design prioritizes generality as the model is trained *without* requiring knowledge on other users.

For our prototype, we apply a reconstruction error based anomaly detection system [181]. Specifically, we use variational autoencoder (VAE) [62], a DNN architecture well-known for *automatically* capturing complex patterns in target data. As shown in Figure 4.5, each VAE starts from an encoder to extract latent features from each input response, followed by a decoder to reconstruct the response from these features. It then computes the mean squared error (MSE) between the input and reconstructed responses, and outputs it as the anomaly score of the input. Ideally, the anomaly score will be low when the input response comes from the legitimate user and high when

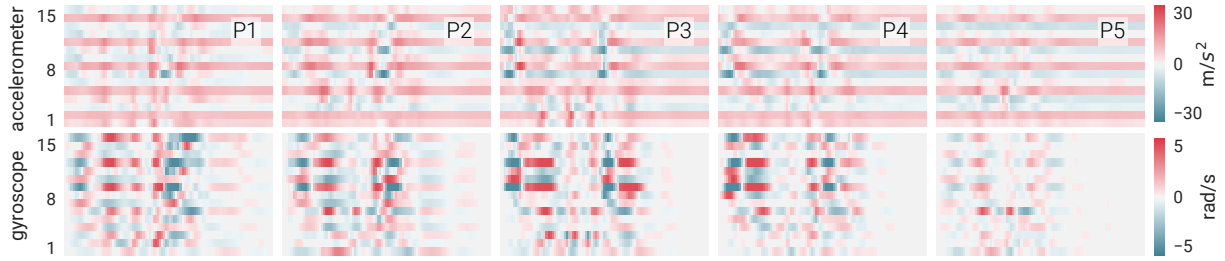

Figure 4.6: Sample responses of a P1's challenge (with L=6 impulses) and impersonators' responses (P2, P3, P4 and P5) to the same challenge. Each row is a sensor channel and each column is one data sample. Here we show one second of responses. When tested on P1's anomaly detection model, the corresponding anomaly scores for P1-5 are 0.70, 5.03, 9.44, 8.81 and 7.50, respectively. In this case, the model can easily detect impersonators.

the input comes from a different user. Thus, the system can configure a threshold on the anomaly score, where a value larger than the threshold indicates the test subject is not the legitimate user (i.e., the user verification fails). In ElectricAuth, we choose the threshold during model training to reach a desired false rejection rate (i.e., the probability that the model rejects the legitimate user's input responses).

For our implementation, we train our VAE using each legitimate user's responses to all the chosen challenges collected during registration. The data aggregation (across challenges) creates a reasonable amount of data to train the VAE successfully. We consider a common VAE architecture [69], where the encoder contains two dense layers of 400 and 200 neurons, respectively, and the decoder contains two dense layers of 400 and 3600 neurons, respectively, to match the input size.

To illustrate the effectiveness of our model, we plot in Figure 4.5b-c the input and reconstructed responses of a legitimate user (here, $P1$ of our user study) and a different participant $P2$ (also from our user study), respectively, using the model trained for $P1$. For the sake of visual clarity, we only plot the responses from only one accelerometer axis. Both responses are not used for model training. We see that $P1$'s response is well-approximated by the model-reconstructed response; in fact, with a very low MSE of 0.057. On the other hand, $P2$'s response (when tested on $P1$'s model), produces a large MSE of 0.604, around 10-fold higher than the MSE of the legitimate user

($P1$).

Figure 4.6 shows the responses (collected by the IMUs) of five subjects (P1-5) to a challenge designed for P1 (the legitimate user in this case). When tested on P1's anomaly detection model, the anomaly scores for these responses are 0.70, 5.03, 9.44, 8.81 and 7.50, respectively. Thus P1's model easily rejects P2-5 as impersonators.

2. Verifying challenge via challenge classifier.

Next in the authentication pipeline, ElectricAuth verifies whether the input response matches the challenge used in the current session. As mentioned earlier, this is designed to resist replay attacks, where an attacker, after obtaining a copy of the legitimate user's responses to previously used challenges, replays one of these responses to bypass authentication.

ElectricAuth implements challenge verification by training a classifier: given an input response, it determines the corresponding challenge. If the identified challenge matches the challenge used in the current authentication session, authentication is granted; otherwise, rejected. Moreover, the classifier also detects when the input response comes from any challenge *not* used to train the classifier, because the classifier will output a low confidence score.

Our implementation uses a Convolutional Neural Network (CNN) for this classification task [174]. It contains four convolutional and two dense layers. Each convolutional layer employs 64 filters sized 5 to extract information from the input. The information is then fed into the two dense layers containing 128 and 112 neurons, respectively. At the end, a softmax function is applied to the output to produce a probability distribution over potential challenges. We train our CNN using the *same* training data used in training the above anomaly detector, except that we now label each response by its corresponding challenge.

4.5 Contributions, Benefits and Limitations

Our main contribution is that we explore EMS in a new direction, i.e., leveraging EMS's *intersubject variability* as a novel modality for active biometric authentication.

ElectricAuth inherits the advantages of both biometric and password authentication: (1) As with any biometric authentication device, ElectricAuth does not require memorization or cognitive effort – this makes our system suited for a wide range of users, including those with cognitive impairments; (2) Unlike passive biometrics (such as fingerprints), ElectricAuth's challenge-response structure makes it secure against data breaches and replay attacks; Lastly, (3) ElectricAuth leverages temporal dependency to create a very large set of challenges – in this way, ElectricAuth can dispose a challenge anytime like a one-time password.

On the flipside, ElectricAuth is subject to several limitations: (1) Like any solution based on electrical muscle stimulation, ElectricAuth requires some initial adjustments of the electrodes (during registration) that ensure pain-free operation, and also periodic re-gelling of adhesive electrodes to prevent electrodes from fatigue and eventually affecting the authentication accuracy; (2) ElectricAuth requires user's hands to be free while authenticating, making it more suitable for hands-free applications; (3) As with existing biometric devices, ElectricAuth requires initial registration. Specifically, each challenge needs to be registered in advance; Lastly, (4) while a single EMS impulse can be very short (e.g., 200μs) to achieve very high accuracy, we expanded our sequence to 1.2 seconds of muscle stimulation, as such ElectricAuth takes $\sim$1300ms to authenticate a user in runtime. While this is certainly fast enough for most applications, it is longer than some passive approaches, such as fingerprint recognition.

4.6 Overview of Evaluations

We evaluated our concept of using EMS for authentication by means of four different evaluations, each shining light on a different facet of our research question. All studies were approved by our Institutional Review Board. To aid the reader in understanding the different validations we performed, we present an overview of our evaluations with a preview of their respective results:

I. User studies. We evaluated the feasibility of EMS as an active biometric with three experiments and 13 participants. We found that that ElectricAuth resists three common attacks: (1)

impersonations attacks, in which participants played impersonators against each legitimate user (attack success rate or false acceptance rate: 0.17%); (2) replay attacks, in which participants mimic the movements of the legitimate user from videos (success rate: 0.00%), or replay a perfect record of response to any used challenges directly into the IMUs (success rate: 0.00%); and, (3) synthesis attacks, in which we synthesized data from the participants' data to attack their authentication models (success rate: 0.2-2.5%).

II. Exploratory longitudinal study. We conducted a longitudinal study over 24 days and for two participants, to examine ElectricAuth's authentication model over time and against various muscle conditions (fatigue, humidity, etc.). We found that an authentication model, trained using the first three days and tested over the next 21 days, performed very stable over time and on muscle conditions unseen during training (false rejection rate $\approx$2%, with a SD around 3%).

III. Technical evaluation. A technical in which we measured ElectricAuth's latency, model training time, and the feasibility of using depth cameras as an alternative motion tracking modality.

IV. Testing model robustness at scale, using synthetic data. We applied a data-driven approach to better understand how our system might scale to larger numbers of users that is simply impractical to test in the laboratory. To realize this, we employed the user study data to train deep generative models that produce synthetic impersonator responses, and used these data to further evaluate ElectricAuth. We found that, across all the data-driven experiments and for all legitimate users, no generated response was accepted by ElectricAuth (attack success rate: 0).

4.7 User Studies

To evaluate the feasibility of EMS as an active biometric we conducted a user study, with three sub-experiments, which allowed us to understand: **(1) authentication accuracy**, in which we evaluated the accuracy of our system; **(2) impersonation attack**, in which we evaluated its robustness against attackers trying to impersonate legitimate users; and, **(3) replay attack and synthesis attack**, in

which we evaluated its robustness against three replay attacks (human mimicry, record-replay, breach-replay) and one online synthesis attack.

In total, we collected 70,000+ wrist and finger movements as responses to EMS challenges (stimulation patterns). We analyzed the performance of ElectricAuth using four standard metrics, typically employed to assess a system's authentication performance: **(1) False rejection rate (FRR)**, which measures how often a legitimate user is mistakenly denied, at a specific threshold; **(2) False acceptance rate (FAR)**, which measures how often an illegitimate user is mistakenly authorized, at a specific threshold; **(3) Equal error rate (EER)**, the rate at which the measured FRR equals the measured FAR for a certain threshold; and, **(4) Receiver operator characteristic curve (ROC curve)**, which describes the relationship between FRR and FAR as a curve, by varying its threshold.

4.7.1 Experiment#1: Authentication Accuracy

The goal of our first study was to understand the authentication accuracy of our system. Furthermore, as we were interested in the impact of the length of the EMS challenges on its performance, we recorded participants' movements to three sets of challenges, based on their number of impulses $L = 1, 2, 6$ (referred to as length-1, -2, and -6 challenges, respectively). For each challenge set we stimulated participants' forearms and recorded finger movements using IMUs.

Participants. We recruited 13 participants from our institution (mean age= 24 years, SD= 3 years; mean weight= $66.3\ kg$, SD= $13.3\ kg$; mean height= $171.2\ cm$, SD= $8.2\ cm$; 7 females, 6 males). Participants were compensated with 50 USD for their time.

Apparatus. Participants wore our system on their left forearm. This included the EMS and IMU components, which were fitted by an experimenter. To ensure participants' comfort with EMS, we calibrated it so that all electrode channels operated pain-free. To ensure that all target muscles were correctly stimulated (see Implementation for details), we gradually increased the intensity during calibration, following calibration process similar to [29]. If a participant felt any discomfort

before reaching the target intensity, we moved to another electrode position. To minimize fatigue, participants rested their elbow on a resting base.

After calibration, we recorded each participant's exact electrode locations by making a custom sleeve with marked positions. These 13 sleeves were later used in Experiment #2, where we examined impersonation attacks (i.e., each impersonator wore the sleeve of a legitimate user to attack our authentication system).

During the study, participants did not receive any specific instruction, since we wanted them to react naturally to the EMS impulses.

EMS challenges. The EMS challenges in our study were configured as previously described, i.e., a challenge was comprised of a sequence of single-shot square-impulses with an intensity of $10mA$ and a pulse-width of $200\mu s$; these sequences were of length-1, -2 or -6. In between each pair of impulses we included a time gap. Each gap was one of seven possible durations ($\frac{1}{30}$s, $\frac{2}{30}$s, ..., $\frac{6}{30}$s, $\frac{7}{30}$s); thus, the recording duration of a length-1, -2, and -6 challenges were 0.6s, 0.8s and 1.2s, respectively. While length-1 challenges were collected in this experiment, these were only used for an analysis in Experiment#2 (anomaly detector performance).

Procedure. To test whether ElectricAuth correctly authenticates our 13 participants, we first registered each participant. Our system did this automatically: (1) a participant feels an EMS challenge, (2) their forearm muscles react involuntarily, and (3) our system records the response. We repeated this process 10 times per challenge. These ten responses were shuffled to remove potential sequence effects. These responses were then randomly divided into a training set (eight responses) and a testing set (two responses). Then, our system took these eight responses (for all challenges) and trained the anomaly detector and challenge classifier for each participant. As cross-validation, we repeated this process to produce 10 authentication models per participant and reported the average test results of these models in all our subsequent experiments.

For length-1 and -2 challenges, we tested the full set of challenges (a total of four for length-1 and 112 for length-2). For length-6 challenges, we were forced to test only a subset, since the

full set includes $68,841,472$ challenges, which would be fatiguing for participants. Therefore, we randomly chose 115 challenges from the full set.

In total, each participant performed 2310 trials: 40 trials of the four length-1 challenges (10 repetitions); 1120 trials of the 112 length-2 challenges (10 repetitions); and, 1150 trials of the 115 length-6 challenges subset (10 repetitions).

Results: overall authentication accuracy. We first examine the accuracy of the end-to-end authentication model, which depends on the accuracy of both the anomaly detection model and the challenge classification model. We defined overall accuracy as the probability that a legitimate response successfully passed the two-step authentication. Note that the accuracy is dependent on the anomaly threshold used by ElectricAuth's authentication model. During model training, we configured the threshold to reach a planned false rejection rate (FRR). Note that the threshold is determined using just the training data (without the knowledge of any run-time testing data). Ideally, the run-time measured FRR (i.e., $1-$accuracy) should equal to the planned FRR.

For each of the 13 registered participants, Table 4.1 summarizes the measured FRR (i.e., $=$ $1-$accuracy) aggregating the results across all 115 challenges (of length 6). Here we reported the results for planned FRR of 2% and 5%. We see that the measured FRR closely matched the planned FRR. Across all the participants, the mean measured FRR is 2.4% (SD of 0.4%) and 5.4% (SD of 0.4%), respectively, matching the two planned FRR values (2% and 5%).

participant	planned FRR	
	2%	*5%*
P1	2.3	6.1
P2	2.1	5.5
P3	2.1	4.9
P4	1.7	5.4
P5	2.6	4.8
P6	2.7	6.2
P7	2.5	5.0
P8	2.3	5.5
P9	3.2	5.8
P10	2.2	5.0
P11	2.3	5.0
P12	2.8	5.5
P13	2.6	5.4
AVG(SD)	2.4(0.4)	5.4(0.4)

Table 4.1: The measured false rejection rate (FRR, %) for all registered participants (P1-P13) closely matched the planned FRR. The measured FRR was calculated for each participant using their test responses to 115 length-6 challenges.

Results: challenge classification accuracy. Digging deeper into the accuracy of our system, we turn to evaluate the accuracy of challenge classification model (as it is the main component protecting against replay attacks). Our accuracy findings are depicted in Figure 4.7.

For length-2 challenges (complete set, i.e., 112 of them) the average accuracy is 99.89% (SD=0.19% across users). And for length-6 subset of challenges we found an accuracy of 99.78% (SD=0.50%). These results also show that the challenges (full set of length-2, subset of length-6) are unique across each other.

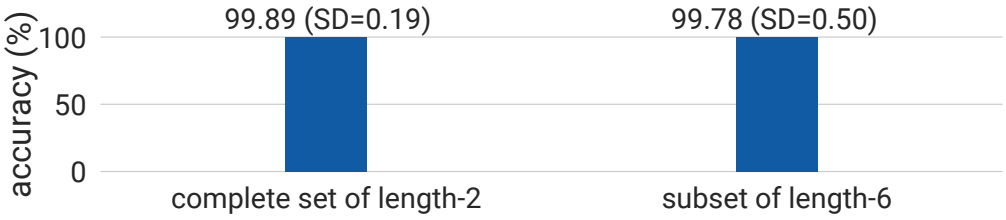

Figure 4.7: ElectricAuth's challenge classification accuracy for length-2 and length-6 challenges.

4.7.2 Experiment#2: Impersonation Attacks

In this user study, we measured our system's ability to resist impersonation attacks.

Participants. For this study, we invited all 13 participants from Study#1. Participants were briefed that they would play an attacker trying to impersonate other participants. Participants were compensated with 50 USD for their time.

Procedure & apparatus. For each target participant, we applied their customized challenges (used in Experiment #1) to the other 12 participants (as impersonators) and collected their responses. Impersonators were asked to wear the sleeve fabricated for each target participant in Experiment #1. These sleeves grant the impersonator with the exact electrode positions of the legitimate user. We also tested cases where impersonators wear their own sleeves and other participants' sleeves and found that wearing the target participant's sleeve leads to the most effective attack; thus we focused on it.

In total, each participant performed 3240 trials: 480 trials of the length-1 challenges (10 repetitions per challenge, impersonated 12 other participants); 2760 trials of the length-6 challenges subset (2 repetitions per challenge, impersonated 12 other participants).

Impersonating someone else by using their electrode placement does not guarantee comfortable use, i.e., we did not adjust electrodes to preserve the legitimate participant's placement. While no participant felt uncomfortable with length-1 challenges, there was some discomfort on a few length-6 trials (3.8% of the total); anytime a participant voiced discomfort, we stopped the stimulation and discarded this trial.

Results: performance of anomaly detector. To deepen our understanding of intersubject variability and the anomaly detection model performance, we first compared the responses to a single stimulus (or length-1 challenge), submitted by each target participant in Experiment#1 and the 12 impersonators in this experiment. We fed these responses to the target participant's anomaly detection model and recorded their anomaly scores. For the sake of visual clarity, we normalized these anomaly score values by the target participant's average anomaly score value (see Figure 4.8).

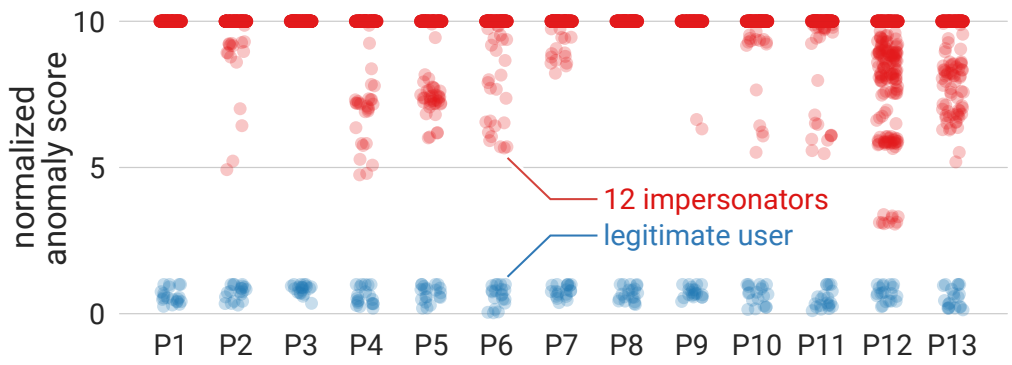

Figure 4.8: Normalized reconstruction error for the responses to each participant's length-1 challenges, submitted by both the legitimate user and the 12 impersonators. For visual clarity, we capped the value at 10.

We found that our anomaly detector for each participant is well-trained and can distinguish impersonators from the legitimate participant. This is clear as Figure 4.8 depicts a large separation between the legitimate participant and the impersonators. It also confirms EMS intersubject

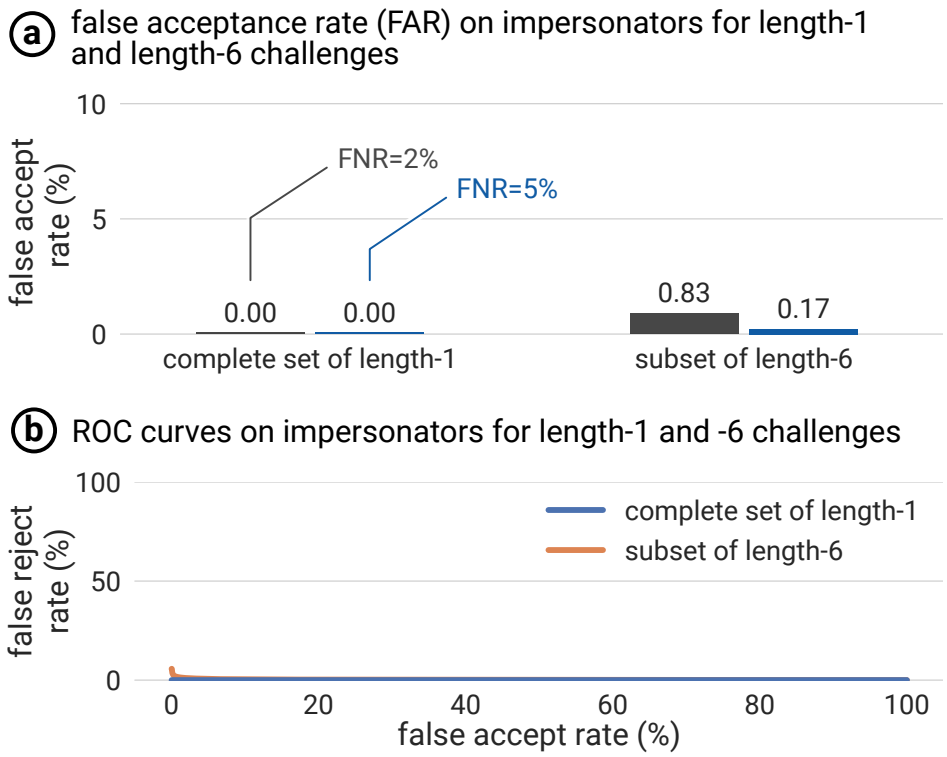

Figure 4.9: ElectricAuth's robustness against impersonation attacks.

variability.

Results: robustness against impersonation attack. We examined the end-to-end success rate of impersonation attacks against each participant, using the attack data collected on length-1 challenges (complete set) and length-6 challenges (the 115 subset).

Figure 4.9(a) depicts the false acceptance rate (aggregated across 13 participants' models since they are consistent) against length-1 and length-6 challenges, for the planned FRR of 2% and 5%, respectively. With length-1 challenges (4 challenges), the impersonation attack failed. With length-6 challenges, the attack exhibited a very low success rate, only 0.83% (SD=1.14%) at planned FRR=2% and 0.17% (SD=0.32%) at planned FRR=5%. Again this suggests that our system is robust against impersonation attacks. Figure 4.9(b) shows the ROC curves under impersonation attacks with length-6 challenges, where ElectricAuth achieves an EER of 1.31%.

4.7.3 Experiment#3: replay and synthesis attacks

In this user study, we measured ElectricAuth's robustness against replay attacks and synthesis attacks, both trying to engineer a response to bypass authentication after obtaining some knowledge on the legitimate user's responses.

We considered three replay attacks, and one synthesis attack, ranging in increased attack complexity:

(1) human mimicry, where the attacker video-tapes and studies a participant's responses and then physically mimics the responses without wearing any EMS;

(2) record-replay, where the attacker compromises the IMUs so that they can *perfectly* record the target participant's response to challenges in previous authentication sessions, then during a new authentication session (i.e., a new challenge), the attacker selects a previous recorded response and directly feeds it to the IMUs;

(3) breach-replay, where the attacker breaches the database or the model to recover stored challenge-response data, and feeds one response to the IMU's circuit; here ElectricAuth reacts to the breach by asking users to re-register using new challenges and retraining the models;

(4) online synthesis, where the attacker compromises both EMS and IMUs to record both the challenge and the response in previous sessions; then at run-time, the attacker searches through these records and attempts to synthesize and submit in *real-time* an engineered response to the current challenge. For these attacks, we evaluated ElectricAuth using the false acceptance rate (FAR) and the ROC curve.

Participants. We recruited five participants to perform the human mimicry attack: three from our previous study (chosen at random) and two new participants from our local institution (ages: 25 & 22 years old; weights: 55 & 99 kg; heights: 177 & 180 cm; one female and one male). Participants were compensated with 50 USD for their time.

Procedure. In the **human mimicry attack**, we asked participants to study 23 videos of finger movements of a target participant. Each video was a recording of one single response to a length-6

challenge. Participants were allowed to study these videos as many times as they intended and in slow-motion (recorded at 240 fps, with clear and unobstructed view of the finger movements). Once confident and ready, participants were asked to mimic these finger movements while wearing only the IMU component of our system, in their best attempt to impersonate the target participant. Furthermore, as reference, we also asked the target participant that had partaken in Experiment#1 to self-mimic 23 of his own EMS responses after observing and studying them.

Results: robustness against human mimicry. We found that none of the study participants was able to fool our system by mimicking the target participant's responses. Note that these participants were allowed to view the videos in slow motion and as many times as they want. The FAR was 0 for a FRR $\geq 2\%$. This confirms our intuition that the EMS movements are indeed involuntary and incredibly hard to voluntarily replicate.

Results: robustness against record-replay attack. For this we utilized data from Experiment#1. Even assuming perfect recording on the side of the attacker (i.e., their recording channel has access to IMUs without any noise or sample rate issues), we found our system to be robust against these attacks. In particular, for length-6 challenges, the FAR (against any of the 115 challenges) was less than 0.0014% across all 13 participants when FRR $\geq 2\%$. This FAR is significantly smaller than the challenge misclassification rate of our authentication model (0.2%, see Experiment#1).

Results: robustness against breach-replay attack. Again we utilized data from Experiment#1. For each participant, we randomly split the 115 challenges (and their responses) into two equal sets (A and B). We assume that the attacker, via data breach, obtains the dataset A and uses them to launch replay attacks against ElectricAuth. At the same time, ElectricAuth reacts to the data breach by asking users to re-register via a set of new challenges (i.e., dataset B) and retraining the authentication models using dataset B. Like the above, we found our system to be robust against these replay attacks – the FAR was less than 0.0098% when FRR $\geq 2\%$. Moreover, both the anomaly detector and challenge classifier components in the model were able to reject the attack

responses.

Results: robustness against online synthesis. We evaluated the success rate of an online **synthesis** attack, using the data from Experiment#1. We assume the attacker has access to the EMS and IMUs without sample rate or noise issues, which is in itself very unlikely. The idea behind a synthesis attack is that the adversary records both challenges and their responses, and segments these into chunks, as in "this impulse at electrode 1, moves this finger by this much", and so forth. We referred to this approach as the simple synthesis attack. A more advanced attack would capture the impact of temporal dependency by segmenting responses into per pair-stimuli chunks, as in "these two impulses at electrodes 1 and 2, move these fingers by this much"). After segmenting the responses, the attacker will observe each incoming impulse of a new challenge and inject a response into the IMUs in real-time. Note that even assuming best hardware and knowledge, assembling this response will always have some latency.

Figure 4.10(a) plots the FAR of online synthesis attacks considering three latency values, assuming the attacker has observed R=50 challenge-response pairs and the planned FRR is set to 5%. Even under the extreme attack case (zero latency, which is physically impossible), the attack success rate is low (i.e., FAR=2.2% and 7.5% for simple and advanced attacks, respectively). When the synthesis latency reaches 20ms, which still depicts an unlikely extremely fast response, the FAR drops to 0.1-0.2%. The same applies when we raised R to 75 (i.e., the advanced attack's success rate is only 0.25% for latency=20ms).

Results: ROC and EER. Finally, Figure 4.10(b) plots the ROC results for all the replay attacks and synthesis attacks (with latency =20ms). We see that ElectricAuth achieves noticeable EERs only for the synthesis attacks (1.48% for simple synthesis and 1.59% for advanced synthesis). These results show that ElectricAuth is robust against replay and synthesis attacks, even those extreme ones.

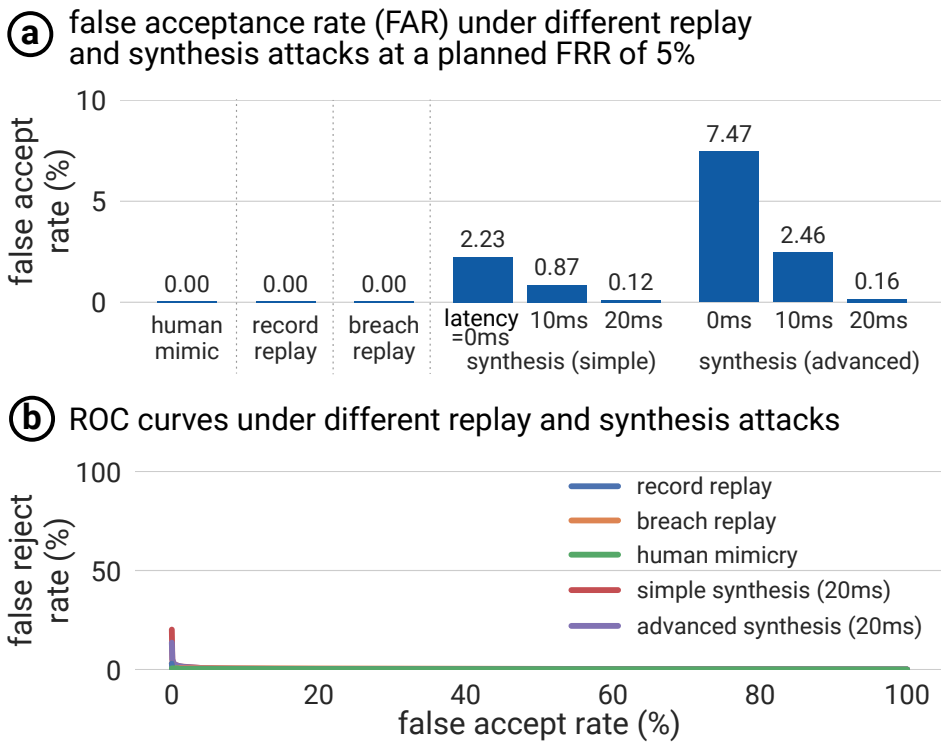

Figure 4.10: ElectricAuth's robustness against different replay and synthesis attacks. For online synthesis, the attacker had perfect records on responses to 50 challenges. Here ElectricAuth operates on length-6 challenges.

4.8 Exploratory longitudinal study

We conducted an exploratory longitudinal study to examine ElectricAuth over time and against various environment and muscle conditions. Specifically, we performed **fixed-model-over-time tests**, which depicts how an authentication model trained using the first three days of data will perform over time and under muscle conditions (e.g., humidity, fatigue, etc.) and other non-predictable environmental factors that were not present in the training data;

Participants. Due to Covid-19, only two co-authors participated in this study (ages: 25 & 24 years old; weights: 70 & $54kg$; heights 170 & $163cm$; one male and one female).

Procedure. In the day prior to the start of the 24-day period, we conducted an initial calibration session (following the same method and apparatus described in Experiment #1). Then, we followed with 24 days of data collection. We collected data once a day. For each participant, we randomly

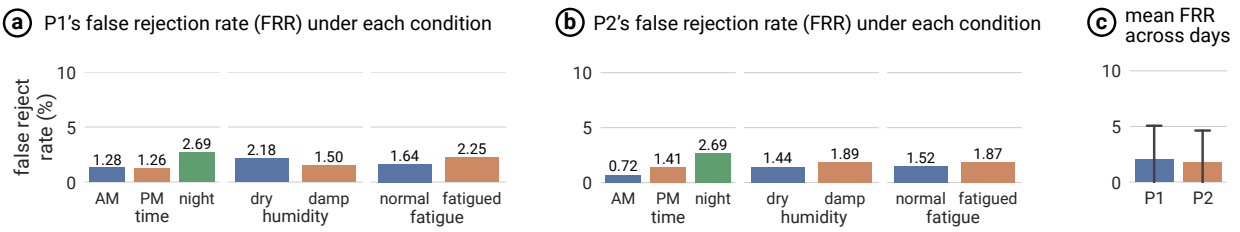

Figure 4.11: Results of fixed-model-over-time tests. (a) and (b) shows for both participants, our system is stable under various conditions; (c) our system is stable over time (21 days) for both participants.

chose 115 length-6 challenges to collect user responses.

For this study, we used fabric sleeves with embedded EMS electrodes at the precalibrated positions for each participant, following a design similar to [110]. Each day, participants were asked to wear their custom electrode-sleeves (depicting their calibrated locations). Participants fitted the sleeve by themselves prior to the trials by aligning markings on the sleeve with their elbow and top of wrist. If the electrode pads were dry, they re-gelled it using conductive gel. Then, they recorded their response to the 115 challenges every day. For each challenge, they collected more than 6 responses per day. After the trials, they removed the sleeve until the next day.

Conditions. To explore the impact of environmental and physiological variations, we conducted data collection under combinations of three conditions: (1) time of the day (morning/afternoon/late); (2) environment humidity (dry/damp); and (3) muscle fatigue (normal/fatigued). We randomly chose one combination per day, and each combination was tested at least twice during the study. In the damp condition, participants were asked to stay in their bathroom with the humidity at over 80% and temperature over 29 $^{\circ}C$ for more than 20 minutes right before the data collection. For dry condition, participants stayed in an air-conditioned room of humidity 55% and temperature $24^{\circ}C$. To test our system right after the muscles started to fatigue, participants were asked to do a routine of intense forearm muscle training (dumbbell wrist flexion and extension) for a minimum of 15 minutes before collecting data.

During the days in which we tested ElectricAuth under normal muscle conditions, participants still performed their forearm muscle training but after the data collection session. This allowed us

to study if extended muscle exercise would affect the system performance.

Training the authentication model. For each participant, we used data collected in the first three days to train the authentication model (the anomaly detector and challenge classifier). For both participants, the training data were collected under the same (dry, pre-workout) condition. The rest of the data (21 days) were used for testing our authentication models. The testing data contained conditions both seen or unseen in the training data. We excluded day 10 and 11 for participant 1 due to need for replenishing the sticky gel on the electrodes, i.e., waiting for gel supply.

For all the trained models, we configured the anomaly detection thresholds to achieve a planned FRR of 2%. As discussed before, such configuration is set using only the training data without the knowledge of any testing data.

Results: fixed-model-over-time tests. To understand the impact of a specific condition (time of the day, environment humidity or muscle fatigue), Fig. 4.11(a)(b) shows the measured false rejection rate (FRR) under each condition. For both participants, the measured FRRs are reasonably consistent across conditions and closely match the planned FRR (2%). But more importantly, while our authentication models are trained only under the combination of dry and pre-workout conditions, they remain accurate under other conditions not seen during training. This provides initial evidence on the generality of ElectricAuth.

For both participants, we also plot the mean FRRs over time in Fig. 4.11(c). We see that the FRR is stable over time, (mean=2.01%, SD=3.13%) for participant 1 and (mean = 1.76%, SD=2.90%) for participant 2. No significant performance degradation over time was found for both participants. These results suggest that ElectricAuth remains relatively stable on a monthly scale.

4.9 Technical evaluation

We deepened our understanding of how future interactive systems might be built based on EMS authentication by measuring system latency, training time, and the feasibility of depth cameras as an alternative tracking modality.

4.9.1 Authentication latency

To measure ElectricAuth's inference latency (i.e., time needed to make a decision in run-time) and the model training, we utilized the data from the participants of Experiment#1, i.e., 115 length-6 challenges, eight response records per challenge for training, two response records for testing.

Run-time inference latency. As we probed the future of EMS-based authentication, we were interested in understanding how ElectricAuth would perform on smaller platforms, such as laptops or even embedded devices. As such, we ran our system on a MacBook Pro with a Intel Core i9-9880H CPU and on a Nvidia Jetson Nano embedded device (measuring 70 x 45 mm). Our results show that our system can authenticate a user in 3ms on laptop's CPU and 35ms on a small embedded device. This result suggests our approach is feasible for quick authentications and even available on mobile or wearable devices.

Training latency. Our results demonstrate that it took 35s (33s for anomaly detector; 2s for the challenge classifier) to train the complete model on a Nvidia Titan RTX GPU and 542s on a laptop's CPU (501s for anomaly detector; 41s for the challenge classifier).

4.9.2 Using camera to capture finger movements

While we used IMUs to capture finger movements in our user study, we believe these movements can also be captured via other modalities, such as depth cameras, a common platform for hand pose estimation [215, 249]. To test our belief, we carried out a simple feasibility experiment. Here, we swapped out IMU sensors with a RGB-D camera (Intel RealSense D435), which operates at

640x480 resolution and 30 frames per second. The camera was placed in front of the participant with a distance of 50cm.

Following the same procedure of Experiment#1, we recorded, via the depth camera, the responses to our 115 length-6 challenges on one participant. We then used an available hand gesture recognition model (from [113]) as our challenge verification model.

We found that the challenge classification accuracy for this simple feasibility experiment was 99.57% using the depth image. We also measure a 0.00% success rate of a record-replay attack against this participant's model.

4.10 Using Synthetic Data to Test Attacks At Scale

Our user study demonstrated that ElectricAuth was accurate in verifying each of the 13 participants and robust against any attacks in that scale. However, gaining insight into how ElectricAuth would perform in larger deployments (e.g., 100's of users) is impractical by means of user studies at an early stage. To shed light into this, we explore a data-driven approach to evaluate ElectricAuth's robustness against impersonation attacks using synthetic data.

Procedure. We followed the recent approach of generating synthetic data by training deep generative models, which is shown to produce diverse and natural data (e.g., objects [202], human faces [275, 11], faces with emotions [139], and physiological data including ECG, EEG, and so forth [88]) beyond the training set. Specifically, we used the PixelCNN++ model [202], a state-of-the-art deep generative model for images (since we treat each response as an image). Following [202], we trained a generative model for each legitimate user in our experiment #2 (see Section 4.7.2), using the impersonator responses collected for this user (12 subjects and 115 challenges), conditioned on the challenge. Once trained, the generator produces random, natural variations of the training data, emulating responses of potential impersonators beyond our user study. We validated each generator using the well-known negative log likelihood (NLL), which produced results on par with (and often slightly better than) those reported by [202] on object/-

face images. This indicated that our trained generators are able to learn and follow the actual data distribution rather than overfitting to the training data.

Results: robustness against synthetic impersonators. For each of the 13 users in our experiment #2, we used the corresponding generator to produce 1075 impersonator responses against this user. These include 100 synthetic impersonators for each of 5 randomly selected challenges, and 5 additional impersonators for each of 115 challenges. We then tested these impersonator responses on ElectricAuth's authentication model for this user (i.e., the same authentication model used in our experiment #2). All impersonator responses were rejected (i.e., 0% FAR at 5% FRR). This result aligns with our user study results, and sheds lights on ElectricAuth's robustness against impersonation attacks at larger scales.

4.11 Conclusions, Applications & Future Work

We proposed, implemented and evaluated the use of electrical muscle stimulation (EMS) as a novel modality for active biometrics. We engineered an interactive system, which we called ElectricAuth, that stimulates the user's forearm muscles with a sequence of electrical impulses (i.e., an EMS challenge) and measures the user's involuntary finger movements (i.e., response to the challenge). The key idea behind ElectricAuth is that it leveraged EMS's *intersubject variability*, i.e., the same electrical stimulation results in different movements in different users because everybody's physiology is unique (e.g., differences in bone and muscular structure, skin resistance and composition, etc.). Moreover, we demonstrated that ElectricAuth is secure against data breaches and replay attacks, as it never reuses the same challenge twice in authentications – the key property that allowed ElectricAuth to achieve this is that in just one second of stimulation our system was able to encode one of 68M possible challenges.

4.11.1 Potential applications

We believe that ElectricAuth is applicable to a range of interactive scenarios in which users authenticate without needing to memorize passwords or PINs. We believe this is of special interest for devices that natively offer motion tracking or finger tracking, such as for virtual reality (which we illustrated in Figure 4.1 using the Oculus Quest), smartwatch-based interaction [141, 247, 276] or even leveraging a smartphone's built in IMUs. Furthermore, we believe our approach is of particular interest for accessibility scenarios, such as authentication for users with motor-impairments (e.g., spinal cord injury, arguably the most impactful application of EMS in the medical domain [178]) but with intact musculature.

4.11.2 Future work

We believe this first exploration of EMS for user authentication provides fertile grounds for exploring subsequent challenges and opportunities: (1) while we have shown ElectricAuth worked well on the full set of 112 length-2 challenges and a subset of 115 length-6 challenges, growing the size of a challenge might enable new applications, as such, research is needed to demonstrate that this approach works across an even larger set of challenges and over a longer time period; (2) while ElectricAuth worked well on the 13 participants from our user studies, more physiological research is needed to deepen understanding of EMS's intersubject variability; (3) while ElectricAuth worked well on the controlled wrist posture, more investigation is required to understand its performance under other postures and their impacts; lastly, (4) as new EMS systems emerge from the medical domain (e.g., higher resolution electrode arrays [106, 110, 184], implanted devices [183], and so forth), a system like ElectricAuth will likely improve in wearability and performance, which will require further investigations.

CHAPTER 5

CONCLUDING THOUGHTS

This dissertation identifies and mitigates new privacy attacks by adapting advanced AI/ML techniques. It first adapts self-supervised learning and self-attention mechanisms to redesign a classic keystroke inference attack, which the security community thought they had successfully addressed. It then identifies a new form of user interface confusion attacks, namely interaction manipulation, for the emerging VR devices. The use of generative AI techniques will significantly amplify the harms of this manipulation attack.

The above findings stress the critical need for building strong defenses against these new attacks. In response, this dissertation also proposes a novel user authentication system designed to counter AI-enhanced biometric spoofing, such as those facilitated by deepfakes of human face or voice. The proposed system leverages tailored autoencoder-based architectures to extract and model fine-grained physiological signatures that are inherently challenging to forge using today's generative models. This design serves as an initial step to establish robust defenses in the face of increasingly sophisticated adversarial techniques.

5.1 Considerations for Future Adaptive Defenses

As AI/ML continues to advance, AI-enhanced privacy attacks will be increasingly sophisticated, underscoring the urgent need for adaptive defenses. Here, the dissertation discusses two directions in this space.

Safeguarding Sensitive Data in the Physical World. Chapter 2 introduces two potent keystroke inference attacks, where an attacker infers a user's sensitive keystroke information from just a small amount of noisy observations. The two attacks provide strong evidence that, as long as an attacker can access some data, they can exploit advanced AI/ML techniques to extract private information from it. Since data is the key ingredient of any AI/ML technique, one obvious defense

is to safeguard user data from unauthorized or adversarial access.

However, safeguarding user data remains an open challenge, particularly in the physical world. This is because users are surrounded by sensors, and they themselves serve as accessible repositories of private information, such as their faces, voices, gestures, typing behaviors, and other physiological or psychological traits.

One promising direction to mitigate data leakage in the physical world would be to disrupt the sensitive data in real-time. For instance, future wearable devices could be designed to dynamically project light patterns onto a user's hands, thereby obstructing vision-based handpose extraction and effectively mitigating the AI-enhanced keystroke inference attack discussed in this dissertation.

Separating Security and Usability. Chapter 3 introduces an interaction manipulation attack, where a remote attacker takes control of a user's interaction with their VR system by trapping the user inside a malicious app that masquerades as the full VR interface. Once trapped, all of the user's interactions with apps, services, and communications with other users can be recorded and modified without their knowledge.

A key lesson learned from this attack is that VR—more broadly, any personal device—is fundamentally a usability-driven system that often overlooks security considerations. VR particularly emphasizes immersive and realistic user experiences, which frequently conflict with conventional security mechanisms such as SSL indicators, multi-factor authentication, and text-based security alerts. In other words, there is a fundamental tension between usability and security that must be carefully addressed.

One potential direction to address this tension is to decouple security and usability mechanisms by deploying them on separate systems. For example, one can design dedicated wearable devices for protecting the security of other devices. The EMS-based authentication discussed in Chapter 4 represents one such approach within this research direction.

REFERENCES

[1] The Logitech K830 keyboard and typing in VR. https://medium.com/xrlo-extended-reality-lowdown/the-logitech-k830-keyboard-and-typing-in-vr-556e2740c48d.

[2] Type in VR with Logitech's keyboard for HTC Vive. https://vrscout.com/news/type-in-vr-logitech-keyboard-kit-for-vive/.

[3] Voice authentication, 2020. https://www.aware.com/voice-authentication/.

[4] ArborXR secures $12m to boost its management platform for AR and VR devices, 2024. https://techcrunch.com/2024/08/13/arborxr-secures-12m-to-boost-management-platform-for-ar-and-vr-devices/.

[5] Meta XR Platform SDK (UPM), 2024. https://developers.meta.com/horizon/downloads/package/meta-xr-platform-sdk/.

[6] Public APIs, 2024. https://github.com/public-apis/public-apis.

[7] Speech-to-speech: Realtime AI voice changer, resemble.ai, 2024. https://www.resemble.ai/voice-changer/.

[8] Ali Ahanger and Anil Kumar. Effect of anthropometric factors on motor nerve conduction velovity in healthy kashmiri population. *International Journal of Medical and Applied Sciences*, 3:125–132, 2014.

[9] Karan Ahuja, Rahul Islam, Varun Parashar, Kuntal Dey, Chris Harrison, and Mayank Goel. Eyespyvr: Interactive eye sensing using off-the-shelf, smartphone-based vr headsets. *In Proc. of IMWUT*, 2(2), 2018.

[10] Kamran Ali, Alex X. Liu, Wei Wang, and Muhammad Shahzad. Keystroke recognition using WiFi signals. In *Proc. of MobiCom*, 2015.

[11] A. Ali-Gombe, E. Elyan, and C. Jayne. Multiple fake classes gan for data augmentation in face image dataset. In *In Proc. of IJCNN*, 2019.

[12] Massih-Reza Amini, Vasilii Feofanov, Loic Pauletto, Emilie Devijver, and Yury Maximov. Self-training: A survey. *arXiv*, 2022.

[13] Android Developers. Build web apps in webview, 2024. `https://developer.android.com/develop/ui/views/layout/webapps/webview`.

[14] Android Developers. Connect to the network, 2024. `https://developer.android.com/develop/connectivity/network-ops/connecting`.

[15] Android Developers. Packagemanager, 2024. `https://developer.android.com/reference/android/content/pm/PackageManager`.

[16] Android Developers. Sending the user to another app, 2024. `https://developer.android.com/training/basics/intents/sending`.

[17] Apple. Vision Pro. `https://www.apple.com/apple-vision-pro/`.

[18] Apple Inc. `https://apps.apple.com/us/app/gboard-the-google-keyboard/id1091700242`.

[19] Abdullah Al Arafat, Zhishan Guo, and Amro Awad. VR-Spy: A side-channel attack on virtual key-logging in vr headsets. In *Proc. of IEEE VR*, 2021.

[20] Eric Arazo, Diego Ortego, Paul Albert, Noel O'Connor, and Kevin McGuinness. Unsupervised label noise modeling and loss correction. In *Proc. of ICML*, 2019.

[21] ArborXR. Device enrollment: Supported methods for enrolling devices into ArborXR, 2024. https://help.arborxr.com/en/collections/3924037-device-enrollment.

[22] Yuki Markus Asano, Christian Rupprecht, and Andrea Vedaldi. Self-labelling via simultaneous clustering and representation learning. In *Proc. of ICLR*, 2020.

[23] D. Asonov and R. Agrawal. Keyboard acoustic emanations. In *Proc. of IEEE S&P*, 2004.

[24] AstricStore. App launcher, 2024. https://assetstore.unity.com/packages/tools/integration/app-launcher-20454.

[25] Abhinaya S. B., Aafaq Sabir, and Anupam Das. Enabling developers, protecting users: Investigating harassment and safety in VR. In *Proc. of USENIX Security*, 2024.

[26] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. Neural machine translation by jointly learning to align and translate. *arXiv*, 2014.

[27] Davide Balzarotti, Marco Cova, and Giovanni Vigna. Clearshot: Eavesdropping on keyboard input from video. In *Proc. of IEEE S&P*, 2008.

[28] Salil P Banerjee and Damon L Woodard. Biometric authentication and identification using keystroke dynamics: A survey. *Journal of Pattern Recognition Research*, 7(1), 2012.

[29] Xueliang Bao, Yuxuan Zhou, Yunlong Wang, Jianjun Zhang, Xiaoying Lü, and Zhigong Wang. Electrode placement on the forearm for selective stimulation of finger extension/flexion. *PloS one*, 13(1), 2018.

[30] Leonard E. Baum. An inequality and associated maximization technique in statistical estimation for probabilistic functions of Markov processes. In *Inequalities III: Proceedings of the Third Symposium on Inequalities*, 1972.

[31] Antonio Bianchi, Jacopo Corbetta, Luca Invernizzi, Yanick Fratantonio, Christopher Kruegel, and Giovanni Vigna. What the app is that? deception and countermeasures in the android user interface. In *Proc. of IEEE S&P*, 2015.

[32] Davide Bove. Sok: The evolution of trusted UI on mobile. In *Proc. of Asia CCS*, 2022.

[33] Doug A. Bowman. Embracing physical keyboards for virtual reality. *Computer*, 53(09):9–10, 2020.

[34] Thomas Brewster. We broke into a bunch of android phones with a 3D-printed head, 2019. `https://www.forbes.com/sites/thomasbrewster/2018/12/13/we-broke-into-a-bunch-of-android-phones-with-a-3d-printed-head/#4a9a79213307`.

[35] Wayne Brown. Perspective projections. `http://learnwebgl.brown37.net/08_projections/projections_perspective.html`.

[36] Saikiran Bulusu, Bhavya Kailkhura, Bo Li, Pramod K. Varshney, and Dawn Song. Anomalous instance detection in deep learning: A survey, 2020.

[37] Daniel Buschek, Alexander De Luca, and Florian Alt. Improving accuracy, applicability and usability of keystroke biometrics on mobile touchscreen devices. In *Proc. of ACM CHI*, 2015.

[38] Abraham G. Campbell, Thomas Holz, John A. Cosgrove, Mike Harlick, and Tadhg O'Sullivan. Uses of virtual reality for communication in financial services: A case study on comparing different telepresence interfaces: Virtual reality compared to video conferencing. *LNNS*, 2019.

[39] Joseph P Campbell. Speaker recognition: A tutorial. *Proceedings of the IEEE*, 85(9):1437–1462, 1997.

[40] Z. Cao, G. Hidalgo Martinez, T. Simon, S. Wei, and Y. A. Sheikh. Openpose: Realtime multi-person 2D pose estimation using part affinity fields. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 2019.

[41] Peter Casey, Ibrahim Baggili, and Ananya Yarramreddy. Immersive virtual reality attacks and the human joystick. *IEEE Transactions on Dependable Secure Computing*, 2021.

[42] Arpan Chakraborty, Brent Harrison, Pu Yang, David Roberts, and Robert St. Amant. Exploring key-level analytics for computational modeling of typing behavior. In *Proc. of HotSoS*, 2014.

[43] R. Chandler, C. Clauser, J. McConville, Herbert Reynolds, and J. Young. Investigation of inertial properties of the human body. page 171, 03 1975.

[44] Theocharis Chatzis, Andreas Stergioulas, Dimitrios Konstantinidis, Kosmas Dimitropoulos, and Petros Daras. A comprehensive study on deep learning-based 3D hand pose estimation methods. *Applied Sciences*, 10(19), 2020.

[45] Bo Chen, Vivek Yenamandra, and Kannan Srinivasan. Tracking keystrokes using wireless signals. *In Proc. of MobiSys*, 2015.

[46] Qi Alfred Chen, Zhiyun Qian, and Z. Morley Mao. Peeking into your app without actually seeing it: UI state inference and novel android attacks. In *Proc. of USENIX Security*, 2014.

[47] Weixuan Chen and Daniel McDuff. Deepphys: Video-based physiological measurement using convolutional attention networks. In *Proc. of ECCV*, 2018.

[48] Kaiming Cheng et al. When the user is inside the user interface: An empirical study of UI security properties in augmented reality. In *Proc. of USENIX Security*, 2024.

[49] Kaiming Cheng, Jeffery F. Tian, Tadayoshi Kohno, and Franziska Roesner. Exploring user

reactions and mental models towards perceptual manipulation attacks in mixed reality. In *Proc. of USENIX Security*, 2023.

[50] Hang Chu, Shugao Ma, Fernando De la Torre, Sanja Fidler, and Yaser Sheikh. Expressive telepresence via modular codec avatars. In *Proc. of ECCV*, 2020.

[51] Charles E Clauser, John T McConville, and J W Young. Weight, volumn, and center of mass of segments of the human body. *Journal of Occupational and Environmental Medicine*, 1971.

[52] William W. Cohen. Enron email dataset, 2015. `https://www.cs.cmu.edu/~enron/`.

[53] The SciPy community. scipy.signal.peak_prominences, 2022. `https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.peak_prominences.html`.

[54] CopyLeaks. Plagiarism checker API - integrate AI powered API, Copyleaks. `https://api.copyleaks.com/`.

[55] Cory Cornelius, Ronald Peterson, Joseph Skinner, Ryan Halter, and David Kotz. A wearable system that knows who wears it. In *Proc. of MobiSys*, 2014.

[56] Philippe De Ryck, Nick Nikiforakis, Lieven Desmet, and Wouter Joosen. Tabshots: client-side detection of tabnabbing attacks. In *Proc. of Asia CCS*, 2013.

[57] A Desplantez, C Cornu, and F Goubel. Viscous properties of human muscle during contraction. *Journal of biomechanics*, 32(6):555–562, 1999.

[58] K.N. Dewangan, G.V. [Prasanna Kumar], P.L. Suja, and M.D. Choudhury. Anthropometric dimensions of farm youth of the north eastern region of india. *International Journal of Industrial Ergonomics*, 35(11):979 – 989, 2005.

[59] Vivek Dhakal. Identification of typing behaviors from large keystroke dataset, 2017. Master Thesis, Aalto University.

[60] Laura Dipietro, Angelo M Sabatini, and Paolo Dario. A survey of glove-based systems and their applications. *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 38(4):461–482, 2008.

[61] C. Doersch and A. Zisserman. Multi-task self-supervised visual learning. In *Proc. of ICCV*, 2017.

[62] Carl Doersch. Tutorial on variational autoencoders, 2016. Preprint arXiv:1606.05908.

[63] R Drillis, R Contini, and M Bluestein. Body segment parameters; a survey of measurement techniques. *Artificial limbs*, 8:44–66, 1964.

[64] Simon Eberz, Nicola Paoletti, Marc Roeschlin, Andrea Patané, Marta Kwiatkowska, and Ivan Martinovic. Broken hearted: How to attack ECG biometrics. In *Proc. of NDSS*, 2017.

[65] EDUCBA. Opencv perspectivetransform. `https://www.educba.com/opencv-perspectivetransform/`.

[66] ElevenLabs. Create a replica of your voice that sounds just like you, 2024. `https://elevenlabs.io/`.

[67] Shangchen Han et al. MEgATrack: Monochrome egocentric articulated hand-tracking for virtual reality. *ACM Transactions on Graphics*, 39, 2020.

[68] Yitan Zhu et al. Converting tabular data into images for deep learning with convolutional neural networks. *Scientific Reports*, 2021.

[69] Basic VAE Example, 2019. `https://github.com/pytorch/examples/tree/master/vae`.

[70] Hugging Face. WER - a hugging face space by evaluate-metric. https://huggingface.co/spaces/evaluate-metric/wer.

[71] Brandon Falk, Yan Meng, Yuxia Zhan, and Haojin Zhu. Poster: Reavatar: Virtual reality de-anonymization attack through correlating movement signatures. In *Proc. of ACM CCS*, 2021.

[72] Farzam Farbiz, Zhou Hao Yu, Corey Manders, and Waqas Ahmad. An electrical muscle stimulation haptic feedback for mixed reality tennis game. In *Proc. of ACM SIGGRAPH*, 2007.

[73] Adeel Faruki et al. Virtual reality as an adjunct to anesthesia in the operating room. *Trials*, 2019.

[74] Anna Maria Feit, Daryl Weir, and Antti Oulasvirta. How we type: Movement strategies and performance in everyday typing. In *Proc. of ACM CHI*, 2016.

[75] Adrienne Porter Felt and David Wagner. Phishing on mobile devices. *Web 2.0 Security and Privacy*, 2011.

[76] Sanja Fidler. Stereo epipolar geometry for general cameras, 2021. http://www.cs.toronto.edu/~fidler/slides/2021Winter/CSC420/lecture13.pdf.

[77] flirtual. The first VR dating app, 2024. https://flirtu.al.

[78] Yanick Fratantonio, Chenxiong Qian, Simon P. Chung, and Wenke Lee. Cloak and dagger: From two permissions to complete control of the ui feedback loop. In *Proc. of IEEE S&P*, 2017.

[79] Martin Georgiev, Subodh Iyengar, Suman Jana, Rishita Anubhai, Dan Boneh, and Vitaly Shmatikov. The most dangerous code in the world: Validating SSL certificates in non-browser software. In *Proc. of ACM CCS*, 2012.

[80] Alberto Giaretta. Security and privacy in virtual reality – a literature survey. *arXiv*, 2022.

[81] Sindhu Reddy Kalathur Gopal, Diksha Shukla, James David Wheelock, and Nitesh Saxena. Hidden reality: Caution, your hand gesture inputs in the immersive virtual world are visible to all! In *Proc. of USENIX Security*, 2023.

[82] Michael J. Grey. Viscoelastic properties of the human wrist during the stabilization phase of a targeted movement. Master's thesis, Simon Fraser University, Burnaby, Canada, December 1997.

[83] Anhong Guo, Robert Xiao, and Chris Harrison. Capauth: Identifying and differentiating user handprints on commodity capacitive touchscreens. In *Proc. of ACM ITS*, 2015.

[84] SDR Harridge, R Bottinelli, M Canepari, MA Pellegrino, C Reggiani, M Esbjörnsson, and B Saltin. Whole-muscle and single-fibre contractile properties and myosin heavy chain isoforms in humans. *Pflügers Archiv*, 432(5):913–920, 1996.

[85] Hasomed, 2020. `https://hasomed.de/en/`.

[86] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proc. of CVPR*, 2016.

[87] Karl Moritz Hermann, Tomas Kocisky, Edward Grefenstette, Lasse Espeholt, Will Kay, Mustafa Suleyman, and Phil Blunsom. Teaching machines to read and comprehend. *In Proc. of NIPS*, 2015.

[88] Andres Hernandez-Matamorosb, Hamido Fujita, and Hector Perez-Meanac. A novel approach to create synthetic biomedical signals using birnn. *Elsevier Information Sciences*, 541:218–241, Dec 2020.

[89] AL Hof. In vivo measurement of the series elasticity release curve of human triceps surae muscle. *Journal of biomechanics*, 31(9):793–800, 1998.

[90] Christian Holz and Patrick Baudisch. Fiberio: A touchscreen that senses fingerprints. In *Proc. of ACM UIST*, 2013.

[91] Christian Holz and Marius Knaust. Biometric touch sensing: Seamlessly augmenting each touch with continuous authentication. In *Proc. of ACM UIST*, 2015.

[92] Jayakumar Hoskere. Everyday AI: Beyond spell check, how google docs is smart enough to correct grammar | google cloud blog. https://cloud.google.com/blog/products/g-suite/everyday-ai-beyond-spell-check-how\-google-docs-is-smart-enough-to-correct-grammar.

[93] Mike Howard. Avatars: The art and science of social presence. https://www.meta.com/blog/quest/avatars-the-art-and-science-of-social-presence/.

[94] Christopher-Eyk Hrabia, Katrin Wolf, and Mathias Wilhelm. Whole hand modeling using 8 wearable sensors: Biomechanics for hand pose prediction. In *Proc. of Augmented Human International Conference*, 2013.

[95] Lin-Shung Huang, Alex Moshchuk, Helen J. Wang, Stuart Schecter, and Collin Jackson. Clickjacking: Attacks and defenses. In *Proc. of USENIX Security*, 2012.

[96] IBM. What is an application programming interface (API)?, 2024. https://www.ibm.com/topics/api.

[97] immersed. Empower work with spatial computing. https://immersed.com.

[98] Report: Data Breach in Biometric Security Platform Affecting Millions of Users, 2019. https://www.vpnmentor.com/blog/report-biostar2-leak/.

[99] Steven A Israel and John M Irvine. Heartbeat biometrics: a sensing system perspective. *International Journal of Cognitive Biometrics*, 1(1):39–65, 2012.

[100] Wolfgang Jank. The em algorithm, its randomized implementation and global optimization: Some challenges and opportunities for operations research. In *Perspectives in operations research*. 2006.

[101] Lasse Jensen and Flemming Konradsen. A review of the use of virtual reality head-mounted displays in education and training. *Education and Information Technologies*, 2018.

[102] Lu Jiang, Zhengyuan Zhou, Thomas Leung, Li-Jia Li, and Li Fei-Fei. Mentornet: Learning data-driven curriculum for very deep neural networks on corrupted labels. In *Proc. of ICML*, 2018.

[103] Xinhui Jiang, Jussi P.P. Jokinen, Antti Oulasvirta, and Xiangshi Ren. Learning to type with mobile keyboards: Findings with a randomized keyboard. *Comput. Hum. Behav.*, 126, jan 2022.

[104] Wenqiang Jin, Srinivasan Murali, Huadi Zhu, and Ming Li. Periscope: A keystroke inference attack using human coupled electromagnetic emanations. In *Proc. of ACM CCS*, 2021.

[105] Zach Jorgensen and Ting Yu. On mouse dynamics as a behavioral biometric for authentication. In *Proc. of ACM CCS*, 2011.

[106] T. Keller, M. Lawrence, A. Kuhn, and M. Morari. New multi-channel transcutaneous electrical stimulation technology for rehabilitation. In *International Conference of the IEEE Engineering in Medicine and Biology Society*, 2006.

[107] David Kim, Otmar Hilliges, Shahram Izadi, Alex D. Butler, Jiawen Chen, Iason Oikonomidis, and Patrick Olivier. Digits: Freehand 3D interactions anywhere using a wrist-worn gloveless sensor. In *Proc. of ACM UIST*, 2012.

[108] Tomi Kinnunen, Md. Sahidullah, Héctor Delgado, Massimiliano Todisco, Nicholas Evans,

Junichi Yamagishi, and Kong Aik Lee. The asvspoof 2017 challenge: Assessing the limits of replay spoofing attack detection. In *Proc. Interspeech 2017*, pages 2–6, 2017.

[109] kiraio-moe. Remove unity splash screen, 2024. `https://github.com/kiraio-moe/remove-unity-splash-screen`.

[110] Jarrod Knibbe, Paul Strohmeier, Sebastian Boring, and Kasper Hornbæk. Automatic calibration of high density electric muscle stimulation. *Proc. of ACM IMWUT*, 2017.

[111] Michinari Kono, Takumi Takahashi, Hiromi Nakamura, Takashi Miyaki, and Jun Rekimoto. Design guideline for developing safe systems that apply electricity to the human body. *ACM Transactions on Computer-Human Interaction*, 25(3), 2018.

[112] Okan Köpüklü, Ahmet Gunduz, Neslihan Kose, and Gerhard Rigoll. Real-time hand gesture detection and classification using convolutional neural networks. In *Proc. of IEEE FG 2019*.

[113] Okan Köpüklü, Ahmet Gunduz, Neslihan Kose, and Gerhard Rigoll. Real-time hand gesture detection and classification using convolutional neural networks. In *IEEE International Conference on Automatic Face & Gesture Recognition*, 2019.

[114] Ernst Kruijff, Dieter Schmalstieg, and Steffi Beckhaus. Using neuromuscular electrical stimulation for pseudo-haptic feedback. In *Proc. of VRST*, 2006.

[115] Dominik Kulon, Riza Alp Guler, Iasonas Kokkinos, Michael M Bronstein, and Stefanos Zafeiriou. Weakly-supervised mesh-convolutional hand reconstruction in the wild. In *Proc. of CVPR*, 2020.

[116] Coding Labs. World, view and projection transformation matrices. `http://www.codinglabs.net/article_world_view_projection_matrix.aspx`.

[117] Vladimir I. Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. *Soviet physics. Doklady*, 10:707–710, 1965.

[118] Jingjie Li, Kassem Fawaz, and Younghyun Kim. Velody: Nonlinear vibration challenge-response for resilient user authentication. In *Proc. of ACM CCS*, 2019.

[119] Mengcheng Li, Liang An, Hongwen Zhang, Lianpeng Wu, Feng Chen, Tao Yu, and Yebin Liu. Interacting attention graph for single image two-hand reconstruction. In *Proc. of CVPR*, 2022.

[120] Mengyuan Li, Yan Meng, Junyi Liu, Haojin Zhu, Xiaohui Liang, Yao Liu, and Na Ruan. When CSI meets public WiFi: Inferring your mobile phone password via WiFi signals. In *Proc. of ACM CCS*, 2016.

[121] Chong L. Lim, Chris Rennie, Robert J. Barry, Homayoun Bahramali, Ilario Lazzaro, Barry Manor, and Evian Gordon. Decomposing skin conductance into tonic and phasic components. *International Journal of Psychophysiology*, 25(2):97 – 109, 1997.

[122] John Lim, True Price, Fabian Monrose, and Jan-Michael Frahm. Revisiting the threat space for vision-based keystroke inference attacks. In *Proc. of ECCV*, 2020.

[123] Katleho Limakatso. How virtual and augmented reality are changing healthcare, 2023. `https://healthnews.com/news/virtual-and-augmented-reality-boom-in-healthcare/`.

[124] Feng Lin, Kun Woo Cho, Chen Song, Wenyao Xu, and Zhanpeng Jin. Brain password: A secure and truly cancelable brain biometrics for smart headwear. In *Proc. of ACM MobiSys*, 2018.

[125] Kang Ling, Yuntang Liu, Ke Sun, Wei Wang, Lei Xie, and Qing Gu. Spidermon: Towards using cell towers as illuminating sources for keystroke monitoring. In *Proc. of IEEE INFOCOM*, 2020.

[126] Zhen Ling, Zupei Li, Chen Chen, Junzhou Luo, Wei Yu, and Xinwen Fu. I know what you enter on Gear VR. In *Proc. of CNS*, 2019.

[127] Zhen Ling, Zupei Li, Chen Chen, Junzhou Luo, Wei Yu, and Xinwen Fu. I know what you enter on Gear VR. In *Proc. of IEEE CNS*, 2019.

[128] LlamaLab. Automate, 2024. `https://llamalab.com/automate/`.

[129] Pedro Lopes and Patrick Baudisch. Muscle-propelled force feedback: Bringing force feedback to mobile devices. In *Proc. of ACM CHI*, 2013.

[130] Pedro Lopes, Alexandra Ion, and Patrick Baudisch. Impacto: Simulating physical impact by combining tactile stimulation with electrical muscle stimulation. In *Proc. of ACM UIST*, 2015.

[131] Pedro Lopes, Alexandra Ion, Willi Mueller, Daniel Hoffmann, Patrik Jonell, and Patrick Baudisch. Proprioceptive interaction. In *Proc. of ACM CHI*, 2015.

[132] Pedro Lopes, Patrik Jonell, and Patrick Baudisch. Affordance++: Allowing objects to communicate dynamic use. In *Proc. of ACM CHI*, 2015.

[133] Pedro Lopes, Sijing You, Lung-Pan Cheng, Sebastian Marwecki, and Patrick Baudisch. Providing haptics to walls & heavy objects in virtual reality by means of electrical muscle stimulation. In *Proc. of ACM CHI*, 2017.

[134] Pedro Lopes, Sijing You, Alexandra Ion, and Patrick Baudisch. Adding force feedback to mixed reality experiences and games using electrical muscle stimulation. In *Proc. of ACM CHI*, 2018.

[135] Pedro Lopes, Doăa Yüksel, François Guimbretière, and Patrick Baudisch. Muscle-plotter: An interactive system based on electrical muscle stimulation that produces spatial output. In *Proc. of ACM UIST*, 2016.

[136] Shiqing Luo, Xinyu Hu, and Zhisheng Yan. Holologger: Keystroke inference on mixed reality head mounted displays. In *Proc. of IEEE VR*, 2022.

[137] Shiqing Luo, Xinyu Hu, and Zhisheng Yan. Holologger: Keystroke inference on mixed reality head mounted displays. In *Proc. of IEEE VR*, 2022.

[138] Shiqing Luo, Anh Nguyen, Hafsa Farooq, Kun Sun, and Zhisheng Yan. Eavesdropping on controller acoustic emanation for keystroke inference attack in virtual reality. In *Proc. of NDSS*, 2024.

[139] Y. Luo and B. Lu. EEG data augmentation for emotion recognition using a conditional wasserstein gan. In *International Conference of the IEEE Engineering in Medicine and Biology Society*, 2018.

[140] Davide Maltoni, Dario Maio, Anil K Jain, and Salil Prabhakar. *Handbook of fingerprint recognition*. Springer Science & Business Media, 2009.

[141] Meethu Malu, Pramod Chundury, and Leah Findlater. Exploring accessible smartwatch interactions for people with upper body motor impairments. In *Proc. of ACM CHI*, 2018.

[142] ManageXR. Preparing a meta quest device for registration, 2024. `https://help.manage xr.com/en/articles/5345953-preparing-a-meta-quest-device-for-registration`.

[143] Philip Marquardt, Arunabh Verma, Henry Carter, and Patrick Traynor. (Sp)IPhone: Decoding vibrations from nearby keyboards using mobile phone accelerometers. In *Proc. of ACM CCS*, 2011.

[144] SMM Martens, Joris M Mooij, N Jeremy Hill, Jason Farquhar, and Bernhard Schölkopf. A graphical model framework for decoding in the visual ERP-Based BCI speller. *Neural Computation*, 23(1):160–182, 01 2011.

[145] Joanna Materzynska, Guillaume Berger, Ingo Bax, and Roland Memisevic. The jester dataset: A large-scale video dataset of human gestures. In *Proc. of IEEE/CVF ICCVW*, 2019.

[146] Joanna Materzynska, Guillaume Berger, Ingo Bax, and Roland Memisevic. The jester dataset: A large-scale video dataset of human gestures. In *Proc. of ICCV Workshops*, 2019.

[147] MediaPipe Hands. Javascript solution api. https://google.github.io/mediapipe/solutions/hands#javascript-solution-api.

[148] Manuel Meier, Paul Streli, Andreas Fender, and Christian Holz. TapID: Rapid touch interaction in virtual reality using wearable sensing. In *Proc. of IEEE VR*, 2021.

[149] Jeanne C. Meister. How companies are using VR to develop employees' soft skills. *Harvard Business Review*, 2021.

[150] Meta. Meta Horizon Workrooms. https://forwork.meta.com/horizon-workrooms.

[151] Meta. Build immersive audio experiences with audio SDK, 2024. https://developer.oculus.com/blog/build-immersive-audio-experiences-meta-quest-sdk/.

[152] Meta. Creativity & Design: Spark stunning creativity with 3D design tools that bring your ideas to life, 2024. https://forwork.meta.com/vr-creativity-design.

[153] Meta. Expand your world with Meta Quest, 2024. https://www.meta.com/quest/.

[154] Meta. Introducing our open mixed reality ecosystem, 2024. https://about.fb.com/news/2024/04/introducing-our-open-mixed-reality-ecosystem/.

[155] Meta. Meta Quest browser, 2024. https://www.meta.com/experiences/1916519981771802/.

[156] Meta. Meta Quest for Business, 2024. https://forwork.meta.com/quest/business-subscription/.

[157] Meta. Tracked keyboards for Meta Quest, 2024. https://www.meta.com/help/quest/articles/headsets-and-accessories/meta-quest-accessories/tracked-keyboards-meta-quest/.

[158] Meta. Pixel codec avatars, 2025. https://research.facebook.com/publications/pixel-codec-avatars/.

[159] Meta Quest. Interaction SDK overview, 2024. https://developer.oculus.com/documentation/unity/unity-isdk-interaction-sdk-overview/.

[160] Wayne J Millar. Distribution of body weight and height: comparison of estimates based on self-reported and observed measures. *Journal of Epidemiology & Community Health*, 40(4):319–323, 1986.

[161] MIT Technology Review Insights. Augmenting the realities of work, 2023. https://www.technologyreview.com/2023/11/29/1083726/augmenting-the-realities-of-work/.

[162] Franziska Mueller, Florian Bernard, Oleksandr Sotnychenko, Dushyant Mehta, Srinath Sridhar, Dan Casas, and Christian Theobalt. Ganerated hands for real-time 3D hand tracking from monocular rgb. In *Proc. of CVPR*, 2018.

[163] Gonzalo Munilla Garrido, Vivek Nair, and Dawn Song. SoK: Data privacy in virtual reality. *Proc. of PETS*, 2024.

[164] Tahrima Mustafa, Richard Matovu, Abdul Serwadda, and Nicholas Muirhead. Unsure how to authenticate on your vr headset? come on, use your head! In *Proc. of IWSPA*, 2018.

[165] Vivek Nair et al. Unique identification of 50,000+ virtual reality users from head & hand motion data. In *Proc. of USENIX Security*, 2023.

[166] Vivek Nair, Gonzalo Munilla Garrido, Dawn Song, and James O'Brien. Exploring the privacy risks of adversarial VR game design. *Proc. of PETS*, 2023.

[167] Anh Nguyen, Xiaokuan Zhang, and Zhisheng Yan. Penetration vision through virtual reality headsets: Identifying 360-degree videos from head movements. In *Proc. of USENIX Security*, 2024.

[168] Marcus Niemietz. Ui redressing: Attacks and countermeasures revisited. *in CONFidence*, 2011.

[169] Marcus Niemietz and Jörg Schwenk. UI redressing attacks on android devices. *Black Hat Abu Dhabi*, 2012.

[170] Gergana Stefanova Nikolova and Yuli Emilov Toshev. Estimation of male and female body segment parameters of the bulgarian population using a 16-segmental mathematical model. *Journal of Biomechanics*, 40(16):3700–3707, 2007.

[171] Marten Oltrogge, Nicolas Huaman, Sabrina Amft, Yasemin Acar, Michael Backes, and Sascha Fahl. Why Eve and Mallory still love android: Revisiting TLS (in)security in android applications. In *Proc. of USENIX Security*, 2021.

[172] Patrick Howell O'Neill. Data leak exposes unchangeable biometric data of over 1 million people, 2019. `https://www.technologyreview.com/f/614163/data-leak-exposes-unchangeable-biometric-data-of-over-1-million-people/`.

[173] OpenAI. Sora, 2024. `https://openai.com/sora/`.

[174] Francisco Javier Ordóñez and Daniel Roggen. Deep convolutional and lstm recurrent neural networks for multimodal wearable activity recognition. *Sensors (Basel)*, 16(1), Jan 2016.

[175] Marcos Ortega, M.G. Penedo, J. Rouco, N. Barreira, and M.J. Carreira. Personal verification based on extraction and characterisation of retinal feature points. *Journal of Visual Languages & Computing*, 20(2):80–90, 2009.

[176] OWASP. Clickjacking defense cheat sheet, 2024. https://cheatsheetseries.owasp.org/cheatsheets/Clickjacking_Defense_Cheat_Sheet.html.

[177] Shwetak N. Patel, Jeffrey S. Pierce, and Gregory D. Abowd. A gesture-based authentication scheme for untrusted public terminals. In *Proc. of ACM UIST*, 2004.

[178] P. Hunter Peckham and Jayme S. Knutson. Functional electrical stimulation for neuromuscular applications. *Annual Review of Biomedical Engineering*, 7(1):327–360, 2005.

[179] petermg. The ocular migraine: Dev Mode master control program, 2024. https://github.com/petermg/TheOcularMigraineMCP.

[180] Ken Pfeuffer, Matthias J. Geiger, Sarah Prange, Lukas Mecke, Daniel Buschek, and Florian Alt. Behavioural biometrics in vr: Identifying people from body motion and relations in virtual reality. In *Proc. of ACM CHI*, 2019.

[181] Marco A.F. Pimentel, David A. Clifton, Lei Clifton, and Lionel Tarassenko. A review of novelty detection. *Signal Processing*, 99:215 – 249, 2014.

[182] Player Counter. VRChat player count and statistics 2023, 2024. https://playercounter.com/vrchat/.

[183] D. Popovic, L. L. Baker, and G. E. Loeb. Recruitment and comfort of bion implanted electrical stimulation: Implications for fes applications. *IEEE Transactions on Neural Systems and Rehabilitation Engineering*, 15(4):577–586, 2007.

[184] Ana Popović-Bijelić, Goran Bijelić, Nikola Jorgovanović, Dubravka Bojanić, Mirjana B.

Popović, and Dejan B. Popović. Multi-field surface electrode for selective electrical stimulation. *Artificial Organs*, 29(6):448–452, 2005.

[185] Luis Prada. Judges are using virtual reality headsets in the courtroom, 2025. https://www.vice.com/en/article/judges-are-using-virtual-reality-headsets-in-the-courtroom/.

[186] PwC. PwC 2022 us metaverse survey, 2022. https://www.pwc.com/us/en/tech-effect/emerging-tech/metaverse-survey.html.

[187] Lawrence Rabiner and Biinghwang Juang. An introduction to hidden markov models. *IEEE Acoustics, Speech, and Signal Processing magazine*, 3(1):4–16, 1986.

[188] Jaziar Radianti, Tim A. Majchrzak, Jennifer Fromm, and Isabell Wohlgenannt. A systematic review of immersive virtual reality applications for higher education: Design elements, lessons learned, and research agenda. *Computers & Education*, 2020.

[189] Rahul Raguram, Andrew M. White, Dibyendusekhar Goswami, Fabian Monrose, and Jan-Michael Frahm. ISpy: Automatic reconstruction of typed input from compromising reflections. In *Proc. of ACM CCS*, 2011.

[190] Vijay Rajanna, Seth Polsley, Paul Taele, and Tracy Hammond. A gaze gesture-based user authentication system to counter shoulder-surfing attacks. In *Proc. of ACM CHI EA*, 2017.

[191] Kasper Bonne Rasmussen, Marc Roeschlin, Ivan Martinovic, and Gene Tsudik. Authentication using pulse-response biometrics. In *Proc. of NDSS*.

[192] Brian Reed. The physiology of neuromuscular electrical stimulation. *Pediatric Physical Therapy*, 9(3):96–102, 1997.

[193] Scott Reed, Honglak Lee, Dragomir Anguelov, Christian Szegedy, Dumitru Erhan, and An-

drew Rabinovich. Training deep neural networks on noisy labels with bootstrapping. *arXiv preprint arXiv:1412.6596*, 2014.

[194] Chuangang Ren, Peng Liu, and Sencun Zhu. Windowguard: Systematic protection of gui security in android. In *Proc. of NDSS*, 2017.

[195] Chuangang Ren, Peng Liu, and Sencun Zhu. Windowguard: Systematic protection of GUI security in android. In *Proc. of NDSS*, 2017.

[196] Chuangang Ren, Yulong Zhang, Hui Xue, Tao Wei, and Peng Liu. Towards discovering and understanding task hijacking in android. In *Proc. of USENIX Security*, 2015.

[197] RikkaApps. Shizuku, 2024. `https://shizuku.rikka.app/guide/setup/`.

[198] Mohd Sabra, Anindya Maiti, and Murtuza Jadliwala. Zoom on the keystrokes: Exploiting video calls for keystroke inference attacks. *CoRR*, abs/2010.12078, 2020.

[199] Mohd Sabra, Nisha Vinayaga Sureshkanth, Ari Sharma, Anindya Maiti, and Murtuza Jadliwala. Exploiting out-of-band motion sensor data to de-anonymize virtual reality users. *Proc. of WiSec*, 2023.

[200] Davide Salanitri, Glyn Lawson, and Brian Waterfield. The relationship between presence and trust in virtual reality. In *Proc. of the European Conference on Cognitive Ergonomics*, 2016.

[201] Johnny Saldaña. The coding manual for qualitative researchers. 2021.

[202] Tim Salimans, Andrej Karpathy, Xi Chen, and Diederik P. Kingma. Pixelcnn++: A pixelcnn implementation with discretized logistic mixture likelihood and other modifications. In *ICLR*, 2017.

[203] Munehiko Sato, Rohan S. Puri, Alex Olwal, Yosuke Ushigome, Lukas Franciszkiewicz, Deepak Chandra, Ivan Poupyrev, and Ramesh Raskar. Zensei: Embedded, multi-electrode bioimpedance sensing for implicit, ubiquitous user recognition. In *Proc. of ACM CHI*, 2017.

[204] Tim Schneeberger. awesome-shizuku, 2024. `https://github.com/timschneeb/awesome-shizuku`.

[205] Joachim Schuster. Check the computer's RSA key fingerprint, 2024. `https://joachimschuster.de/posts/debug-on-device-rsa-fingerprint/`.

[206] SeleniumHQ. SeleniumHQ/selenium: A browser automation framework and ecosystem. `https://github.com/SeleniumHQ/selenium`.

[207] Alok Sharma, Edwin Vans, Daichi Shigemizu, Keith A. Boroevich, and Tatsuhiko Tsunoda. DeepInsight: A methodology to transform a non-image data to an image for convolution neural network architecture. *Scientific Reports*, 2019.

[208] Lei Shi, Yifan Zhang, Jian Cheng, and Hanqing Lu. Decoupled spatial-temporal attention network for skeleton-based action-gesture recognition. In *Proc. of ACCV*, 2020.

[209] Maliheh Shirvanian and Nitesh Saxena. Wiretapping via mimicry: Short voice imitation man-in-the-middle attacks on crypto phones. In *Proc. of ACM CCS*, 2014.

[210] Diksha Shukla, Rajesh Kumar, Abdul Serwadda, and Vir V. Phoha. Beware, your hands reveal your secrets! In *Proc. of ACM CCS*, 2014.

[211] SideQuest. Sidequest turns 3: New features & 2.2 million monthly active users, 2022. `https://www.uploadvr.com/sidequest-turns-3-sponsored/`.

[212] SideQuest. Custom home, 2024. `https://sidequestvr.com/apps/customhome/0/rating`.

[213] T. Simon, H. Joo, I. Matthews, and Y. Sheikh. Hand keypoint detection in single images using multiview bootstrapping. In *Proc. of CVPR*, 2017.

[214] Tomas Simon, Hanbyul Joo, Iain Matthews, and Yaser Sheikh. Hand keypoint detection in single images using multiview bootstrapping. In *Proc. of CVPR*, 2017.

[215] Ayan Sinha, Chiho Choi, and Karthik Ramani. Deephand: Robust hand pose estimation by completing a matrix imputed with deep features. In *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 4150–4158, June 2016.

[216] Bill Siwicki. What the metaverse and virtual reality can contribute to healthcare, 2023. `https://www.healthcareitnews.com/news/what-metaverse-and-virtual-reality-can-contribute-healthcare`.

[217] Carter Slocum, Yicheng Zhang, Nael Abu-Ghazaleh, and Jiasi Chen. Going through the motions: AR/VR keylogging from user head motions. In *Proc. of USENIX Security*, 2023.

[218] Carter Slocum, Yicheng Zhang, Erfan Shayegani, Pedram Zaree, Nael Abu-Ghazaleh, and Jiasi Chen. That doesn't go there: Attacks on shared state in Multi-User augmented reality applications. In *Proc. of USENIX Security*, 2024.

[219] Ivo Sluganovic, Marc Roeschlin, Kasper B. Rasmussen, and Ivan Martinovic. Using reflexive eye movements for fast challenge-response authentication. In *Proc. of ACM CCS*, 2016.

[220] Dawn Xiaodong Song, David Wagner, and Xuqing Tian. Timing analysis of keystrokes and timing attacks on SSH. In *Proc. of USENIX Security Symposium*, 2001.

[221] Hwanjun Song, Minseok Kim, Dongmin Park, and Jae-Gil Lee. Learning from noisy labels with deep neural networks: A survey. *CoRR*, abs/2007.08199, 2020.

[222] Spiceworks. Spiceworks study reveals nearly 90 percent of businesses will use biometric authentication technology by 2020, 2018. https://www.spiceworks.com/press/releases/spiceworks-study-reveals-nearly-90-percent-businesses-will-use-biometric-authentication-technology-2020/.

[223] Diana S. Stetson, James W. Albers, Barbara A. Silverstein, and Robert A. Wolfe. Effects of age, sex, and anthropometric factors on nerve conduction measures. *Muscle & Nerve*, 15(10):1095–1104, 1992.

[224] Primož Strojnik, Alojz Kralj, and I Ursic. Programmed six-channel electrical stimulator for complex stimulation of leg muscles during walking. *IEEE Transactions on Biomedical Engineering*, (2):112–116, 1979.

[225] Zihao Su et al. Remote keylogging attacks in multi-user VR applications. In *Proc. of USENIX Security*, 2024.

[226] Maki Sugimoto. *Cloud XR (Extended Reality: Virtual Reality, Augmented Reality, Mixed Reality) and 5G Mobile Communication System for Medical Image-Guided Holographic Surgery and Telemedicine*. 2022.

[227] Fangmin Sun, Chenfei Mao, Xiaomao Fan, and Ye Li. Accelerometer-based speed-adaptive gait authentication method for wearable iot devices. *IEEE Internet of Things Journal*, 6(1):820–830, 2019.

[228] Fangmin Sun, Chenfei Mao, Xiaomao Fan, and Ye Li. Accelerometer-based speed-adaptive gait authentication method for wearable iot devices. *IEEE Internet of Things Journal*, 6(1):820–830, 2019.

[229] Emi Tamaki, Takashi Miyaki, and Jun Rekimoto. Possessedhand: Techniques for controlling human hands using electrical muscles stimuli. In *Proc. of ACM CHI*, 2011.

[230] Pin Shen Teh, Andrew Beng Jin Teoh, and Shigang Yue. A survey of keystroke dynamics biometrics. *The Scientific World Journal*, 2013, 2013.

[231] Wen-Jie Tseng, Elise Bonnail, Mark McGill, Mohamed Khamis, Eric Lecolinet, Samuel Huron, and Jan Gugenheimer. The dark side of perceptual manipulations in virtual reality. In *Proc. of ACM CHI*, 2022.

[232] The Muscle Physiology Laboratory UCSD. Fundamental functional properties of skeletal muscle, 2008. `http://muscle.ucsd.edu/musintro/props.shtml`.

[233] Kai Uffmann, Stefan Maderwald, Waleed Ajaj, Craig G Galban, Serban Mateiescu, Harald H Quick, and Mark E Ladd. In vivo elasticity measurements of extremity skeletal muscle with mr elastography. *NMR in Biomedicine: An International Journal Devoted to the Development and Application of Magnetic Resonance In Vivo*, 17(4):181–190, 2004.

[234] Ülkü Meteriz-Yildiran, Necip Fazil Yildiran, Amro Awad, and David Mohaisen. A keylogging inference attack on air-tapping keyboards in virtual environments. In *Proc. of IEEE VR*, 2022.

[235] Unity. Unity UI: Unity user interface, 2024. `https://docs.unity3d.com/Packages/com.unity.ugui@1.0/manual/index.html`.

[236] University of Notre Dame. The frequency of the letters of the alphabet in english. `https://www3.nd.edu/~busiforc/handouts/cryptography/letterfrequencies.html`.

[237] Valentin Bazarevsky and Fan Zhang. On-Device, Real-Time Hand Tracking with MediaPipe, 2021. `https://ai.googleblog.com/2019/08/on-device-real-time-hand-tracking-with.html`.

[238] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N.

Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Proc. of NeurIPS*, 2017.

[239] Verizon. 2023 mobile security index white paper. 11 2023.

[240] Andrew Viterbi. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE transactions on Information Theory*, 13(2):260–269, 1967.

[241] VOICE.AI. Real-time voice cloning software, 2024. `https://voice.ai/hub/app/real-time-voice-cloning-software/`.

[242] Jan-Niklas Voigt-Antons, Tanja Kojic, Danish Ali, and Sebastian Möller. Influence of hand tracking as a way of interaction in virtual reality on user experience. In *Proc. of QoMEX*, 2020.

[243] Martin Vondráček, Ibrahim Baggili, and Peter Casey. Rise of the metaverse's immersive virtual reality malware and the man-in-the-room attack & defenses. *Computers & Security*, 2022.

[244] Oculus VR. Stereo-based calibration apparatus. `https://patents.justia.com/patent/9805512`.

[245] VRChat. Create, share, play, 2024. `https://hello.vrchat.com`.

[246] vSpatial. The workspace of the future is here. `https://www.vspatial.com/xr`.

[247] Tran Huy Vu, Archan Misra, Quentin Roy, Kenny Choo Tsu Wei, and Youngki Lee. Smartwatch-based early gesture detection 8 trajectory tracking for interactive gesture-driven applications. *Proc. of ACM IMWUT*, 2(1), 2018.

[248] Vuplex. 3D webview: the ultimate cross-platform web browser for unity, 2024. `https://developer.vuplex.com/webview/overview`.

[249] Chengde Wan, Thomas Probst, Luc Van Gool, and Angela Yao. Self-supervised 3D hand pose estimation through training by fitting. In *Proc. of CVPR*, 2019.

[250] He Wang, Ted Tsung-Te Lai, and Romit Roy Choudhury. MoLe: Motion leaks through smartwatch sensors. In *Proc. of MobiCom*, 2015.

[251] Jiayi Wang et al. RGB2Hands: Real-time tracking of 3D hand interactions from monocular RGB video. *ACM Trans. Graph.*, nov 2020.

[252] Richard P Wildes. Iris recognition: an emerging biometric technology. *Proceedings of the IEEE*, 85(9):1348–1363, 1997.

[253] Davey Winder. Apple's iphone faceid hacked in less than 120 seconds, 2019. `https://www.forbes.com/sites/daveywinder/2019/08/10/apples-iphone-faceid-hacked-in-less-than-120-seconds/#449ff4b621bc`.

[254] Davey Winder. Hackers claim any smartphone fingerprint lock can be broken in 20 minutes, 2019. `https://www.forbes.com/sites/daveywinder/2019/11/02/smartphone-security-alert-as-hackers-claim-any-fingerprint-lock-broken-in-20-minutes/#588a90116853`.

[255] Saining Xie, Ross Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. In *Proc. of CVPR*, 2017.

[256] Saining Xie, Ross B. Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. *CoRR*, abs/1611.05431, 2016.

[257] BRINK XR. Explore the world, 2025. `https://www.brinkxr.com`.

[258] Yi Xu, Jared Heinly, Andrew M White, Fabian Monrose, and Jan-Michael Frahm. Seeing double: Reconstructing obscured typed input from repeated compromising reflections. In *Proc. of ACM CCS*, 2013.

[259] Yi Xu, True Price, Jan-Michael Frahm, and Fabian Monrose. Virtual U: defeating face liveness detection by building virtual models from your public photos. In Thorsten Holz and Stefan Savage, editors, *Proc. of USENIX Security*, 2016.

[260] Edwin Yang, Song Fang, IanMarkwood, Yao Liu, Shangqing Zhao, Zhuo Lu, and Haojin Zhu. Wireless training-free keystroke inference attack and defense. *IEEE/ACM Transactions on Networking*, 30(4):1733–1748, 2022.

[261] Zhuolin Yang, Yuxin Chen, Zain Sarwar, Hadleigh Schwartz, Ben Y. Zhao, and Haitao Zheng. Towards a general video-based keystroke inference attack. In *Proc. of USENIX Security Symposium*, 2023.

[262] Zhuolin Yang, Zain Sarwar, Iris Hwang, Ronik Bhaskar, Ben Y. Zhao, and Haitao Zheng. Can virtual reality protect users from keystroke inference attacks? In *Proc. of USENIX Security*, 2024.

[263] Xin Yi, Chun Yu, Mingrui Zhang, Sida Gao, Ke Sun, and Yuanchun Shi. Atk: Enabling ten-finger freehand typing in air based on 3D hand tracking data. In *Proc of UIST*, 2015.

[264] Xingrui Yu, Bo Han, Jiangchao Yao, Gang Niu, Ivor Tsang, and Masashi Sugiyama. How does disagreement help generalization against label corruption? In *Proc. of ICML*, 2019.

[265] Qinggang Yue, Zhen Ling, Xinwen Fu, Benyuan Liu, Kui Ren, and Wei Zhao. Blind recognition of touched keys on mobile devices. In *Proc. of ACM CCS*, 2014.

[266] Qinggang Yue, Zhen Ling, Wei Yu, Benyuan Liu, and Xinwen Fu. Blind recognition of text input on mobile devices via natural language processing. In *Proc. of PAMCO*, 2015.

[267] Chen Yunfang, Zhu Yihong, Zhou Hao, Chen Wei, and Zhang Wei. Enhanced keystroke recognition based on moving distance of keystrokes through WiFi. In *Proc. of NSS*, 2018.

[268] Baowen Zhang, Yangang Wang, Xiaoming Deng, Yinda Zhang, Ping Tan, Cuixia Ma, and Hongan Wang. Interacting two-hand 3D pose and shape reconstruction from single color image. In *Proc. of ICCV*, 2021.

[269] Fan Zhang, Valentin Bazarevsky, Andrey Vakunov, Andrei Tkachenka, George Sung, Chuo-Ling Chang, and Matthias Grundmann. MediaPipe Hands: On-device real-time hand tracking. *CoRR*, abs/2006.10214, 2020.

[270] Hongyi Zhang, Moustapha Cissé, Yann N. Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimization. *CoRR*, abs/1710.09412, 2017.

[271] Hongyi Zhang, Moustapha Cisse, Yann N. Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimization. *arXiv:1710.09412*, 2017.

[272] Linjun Zhang, Zhun Deng, Kenji Kawaguchi, Amirata Ghorbani, and James Y. Zou. How does mixup help with robustness and generalization? *CoRR*, abs/2010.04819, 2020.

[273] Yicheng Zhang, Carter Slocum, Jiasi Chen, and Nael Abu-Ghazaleh. It's all in your head(set): Side-channel attacks on AR/VR systems. In *Proc. of USENIX Security*, 2023.

[274] Yifan Zhang, Congqi Cao, Jian Cheng, and Hanqing Lu. Egogesture: A new dataset and benchmark for egocentric hand gesture recognition. *IEEE Transactions on Multimedia*, 20(5):1038–1050, 2018.

[275] Jian Zhao, Lin Xiong, Karlekar Jayashree, Jianshu Li, Fang Zhao, Zhecan Wang, Sugiri Pranata, Shengmei Shen, Shuicheng Yan, and Jiashi Feng. Dual-agent gans for photorealistic and identity preserving profile face synthesis. In *Proc. of NeurIPS*, 2017.

[276] Junhan Zhou, Yang Zhang, Gierad Laput, and Chris Harrison. Aurasense: Enabling expressive around-smartwatch interactions with electric field sensing. In *Proc. of ACM UIST*, 2016.

[277] Xiaojin Zhu. Semi-supervised learning literature survey. *University of Wisconsin – Madison*, 2008.

[278] Li Zhuang, Feng Zhou, and J. Tygar. Keyboard acoustic emanations revisited. In *Proc. of ACM CCS*, 2005.