Towards a General Video-based Keystroke Inference Attack

Zhuolin Yang[†]
University of Chicago

Yuxin Chen[†]
University of Chicago

Zain Sarwar
University of Chicago

Hadleigh Schwartz[*]
Columbia University

Ben Y. Zhao
University of Chicago

Haitao Zheng
University of Chicago

Abstract

A large collection of research literature has identified the privacy risks of keystroke inference attacks that use statistical models to extract content typed onto a keyboard. Yet existing attacks cannot operate in realistic settings, and rely on strong assumptions of labeled training data, knowledge of keyboard layout, carefully placed sensors or data from other side-channels. This paper describes experiences developing and evaluating a general, video-based keystroke inference attack that operates in common public settings using a single commodity camera phone, with no pretraining, no keyboard knowledge, no local sensors, and no side-channels. We show that using a self-supervised approach, noisy finger tracking data from a video can be processed, labeled and filtered to train DNN keystroke inference models that operate accurately on the same video. Using IRB approved user studies, we validate attack efficacy across a variety of environments, keyboards, and content, and users with different typing behaviors and abilities. Our project website is located at: https://sandlab.cs.uchicago.edu/keystroke/.

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 [65]. RF-based attacks [33, 59] 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 [28], or zero movement anywhere within 20 meters of Jane [35].

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 [6] or by capturing a view of the screen reflected by her eyeballs [43, 54]. To help train inference models, a recent work produces synthetic data by overlaying a thumb image on mobile keyboards [34]. In contrast,

[†] Both authors contributed equally to this research.
[*] Work done while studying at the University of Chicago.

Figure 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.

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 [57], 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 [47].

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 screen-covers 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 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 [7, 11, 17]. 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 ∼3000 keystrokes, 2 orders of magnitude less than required for general self-supervised vision tools [4, 18]. 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.

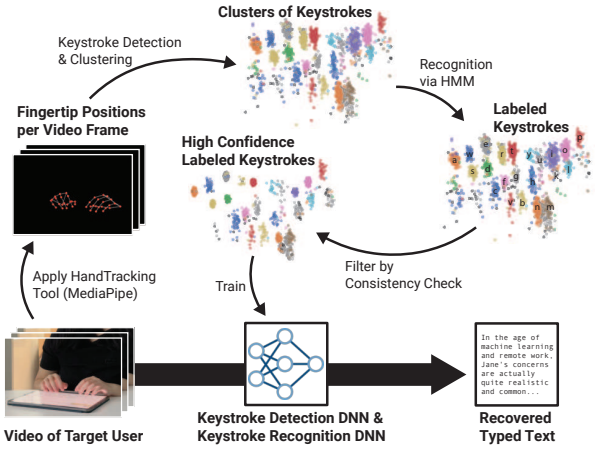

Figure 2: Our self-supervised approach to keystroke inference. We first run unsupervised inference on fingertip data extracted from each video frame, from which we identify keystrokes with high confidence labels (this process is marked by thin arrows). We use these as training data and build DNN models that detect and recognize keystrokes directly from the video (thick arrow).

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

[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.

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 2.

We evaluate this attack using IRB-approved user studies under a variety of conditions, varying 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 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 [27]. 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 [7, 10, 11, 17, 21]. 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 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 [36, 51].

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 [65], and its attack range can be extended to 15m using a bulky parabolic microphone [5]. 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 [37]. 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 [28]. 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 [59]), an attacker could capture the subtle WiFi signal variation caused by 1-finger key entries. One attack [33] 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 [13, 33, 59]. Another study [1] 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 [35]. 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 [6, 34], or by capturing the screen reflected by their eyeballs[2] [43, 54].

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 [57, 58]), 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 [47]. 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 [45]. 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

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

the attacker uses only a frontal view of Jane's typing hands and nothing else (see the threat model in §3).

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 such as MediaPipe.

There are two types of hand tracking tools available today. Depth-based hand tracking [12] 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 [41]. Today's SOTA models provide cm-level accuracy on known poses [30, 32, 52, 60] and 5cm mean error on others [41]. To our knowledge, there is no specialized hand tracking tool for keystroke detection. Prior work on keystrokes [21] 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 [9, 61] (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.

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).

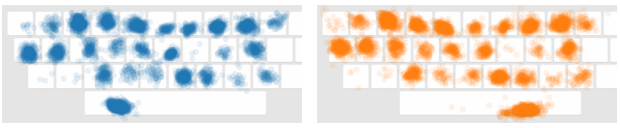

Figure 3: Touchpoints of keystrokes recorded by a touchscreen keyboard, where each circle is a touch point. The separation between neighboring keys' touchpoints is barely $1cm$ in average.

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 [2]), 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.

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.

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 that for most users, the separation between positions of their fingers hitting two neighboring keys, is quite small. Figure 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 [65].

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 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 §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.

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.

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 2). 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 (§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 (§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.

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 [65], 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.

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 [40]. 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 [9], because a keystroke is produced by a fingertip pressing down on the surface. Figure 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.

Figure 4: An example of MediaPipe hand tracking output.

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 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.

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.

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

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

[4] Peak prominence measures how much a peak stands out from the surrounding signal baseline, and can be computed via a SciPy function [15].

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 7 in Appendix). Thus in §5.3, we apply a different method to separate thumb and non-thumb keystrokes.

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 (§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 [55], negative acceleration used in §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`) [19]. 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% [38]. 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 frequency English keys can have more than 5 times amount of samples compared to low-frequency keys [49].

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.

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) [42] to estimate the typed content [65]. 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 [23],

[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.

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 [8] to analyze the cluster data.

Given $\mathbf{T}$ and $\mathbf{E}$, the attacker applies the Viterbi algorithm [50] to infer the most likely hidden state sequence (or typed keys) for the keystroke sequence. Note that unlike [65], 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 [25], 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 §6.1). 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.

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 5 plots, for user PB and the three tracking methods, the estimated touchpoints 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 1 lists the detailed results, from keystroke detection accuracy, clustering accuracy to content accuracy. For

Figure 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 1: Performance of unsupervised inference on handpose data, using three different hand tracking tools.

a fair evaluation, we also list the content accuracy after applying a public spell check tool by Google Docs [24] (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 §7.1), 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.

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 corresponding video 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.

6.1 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 §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.

6.2 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 hand-pose 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 [22]) and apply transfer learning using a public teacher model, which is pre-trained using the EgoGesture dataset [64].

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 §6.1 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 §6.3).

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 [53], a well-known 3D-CNN architecture for video-based classification tasks. The training pipeline is similar to that of gesture recognition [29] – we apply transfer learning from a public teacher model (ResNeXt-101 pre-trained on the Jester dataset [39]). 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 §6.1). 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.

6.3 Noise-aware Model Training

Since the training data for both DNN models can be noisy, we apply three kinds of noise-aware training techniques [3, 48] to suppress their impact on the trained models.

Preventing overfitting. We apply Mixup [62, 63] 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 [3, 26, 48, 56], 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 [44] 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 [3] 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.

7 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 (§7.2);
- **Attack performance across 16 different users**, who have different typing behaviors and abilities (§7.3);
- **Contributions of individual components** (§7.4);
- **Attack complexity** in terms of computing time (§7.5).

7.1 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 [14]. 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 §7.2 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 [40]. 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 [46] 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 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 [31] 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 [20] 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 [16], 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 §7.3, we also study the effectiveness of recovering websites typed during the study, in terms of the **top-k accuracy**.

7.2 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 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%).

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

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 1 (d). Table 2 summarizes the 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 3 summarizes the attack performance, in terms of mean and std for CER, WER, and Similarity, since we run 5 experiments

Figure 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.

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.

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 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 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 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 participant 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 5 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 5 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 5, 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 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 6 summarizes, 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 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.

[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.

7.3 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 [14] ($\approx$ 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].

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 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 §5.5) by placing color tapes on their nails (except for the thumbs) and run our attack.

Overall attack performance. Table 7 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 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%.

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

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

7.4 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 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.

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

Impact of hand tracking tools. To examine the impact of hand tracking tools, we test our attack pipeline by replacing MediaPipe [9, 61] with IntagHand [32], a recently released hand 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.

7.5 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.

8 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.

9 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.

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

[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.

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.

10 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.

Acknowledgements

We thank our anonymous reviewers and shepherd for their insightful feedback. This work is supported in part by NSF grants CNS-1949650, CNS-1923778, and the DARPA GARD program. Opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of any funding agencies.

References

[1] Kamran Ali, Alex X. Liu, Wei Wang, and Muhammad Shahzad. Keystroke recognition using WiFi signals. In *Proc. of MobiCom*, 2015.

[2] Apple Inc. https://apps.apple.com/us/app/gboard-the-google-keyboard/id1091700242.

[3] Eric Arazo, Diego Ortego, Paul Albert, Noel O'Connor, and Kevin McGuinness. Unsupervised label noise modeling and loss correction. In *Proc. of ICML*, 2019.

[4] Yuki Markus Asano, Christian Rupprecht, and Andrea Vedaldi. Self-labelling via simultaneous clustering and representation learning. In *Proc. of ICLR*, 2020.

[5] D. Asonov and R. Agrawal. Keyboard acoustic emanations. In *Proc. of IEEE S&P*, 2004.

[6] Davide Balzarotti, Marco Cova, and Giovanni Vigna. Clearshot: Eavesdropping on keyboard input from video. In *Proc. of IEEE S&P*, 2008.

[7] Salil P Banerjee and Damon L Woodard. Biometric authentication and identification using keystroke dynamics: A survey. *Journal of Pattern Recognition Research*, 7(1), 2012.

[8] 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.

[9] Valentin Bazarevsky and Fan Zhang. On-device, real-time hand tracking with MediaPipe. https://ai.googleblog.com/2019/08/on-device-real-time-hand-tracking-with.html, 2021.

[10] Daniel Buschek, Alexander De Luca, and Florian Alt. Improving accuracy, applicability and usability of keystroke biometrics on mobile touchscreen devices. In *Proc. of CHI*, 2015.

[11] 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.

[12] 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.

[13] Bo Chen, Vivek Yenamandra, and Kannan Srinivasan. Tracking keystrokes using wireless signals. *In Proc. of MobiSys*, 2015.

[14] William W. Cohen. Enron email dataset. https://www.cs.cmu.edu/~enron/, 2015.

[15] The SciPy community. scipy.signal.peak_prominences. https://docs.scipy.org/doc/scipy/reference/generated/scipy.signal.peak_prominences.html, 2022.

[16] CopyLeaks. Plagiarism checker api - integrate ai powered api, copyleaks. https://api.copyleaks.com/.

[17] Vivek Dhakal. Identification of typing behaviors from large keystroke dataset. Master Thesis, Aalto University, 2017.

[18] C. Doersch and A. Zisserman. Multi-task self-supervised visual learning. In *Proc. of ICCV*, 2017.

[19] EDUCBA. Opencv perspectivetransform. https://www.educba.com/opencv-perspectivetransform/.

[20] Hugging Face. WER - a hugging face space by evaluate-metric. https://huggingface.co/spaces/evaluate-metric/wer.

[21] Anna Maria Feit, Daryl Weir, and Antti Oulasvirta. How we type: Movement strategies and performance in everyday typing. In *Proc. of CHI*, 2016.

[22] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proc. of CVPR*, 2016.

[23] 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.

[24] 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.

[25] Wolfgang Jank. The em algorithm, its randomized implementation and global optimization: Some challenges and opportunities for operations research. In *Perspectives in operations research*. 2006.

[26] 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.

[27] 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.

[28] 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.

[29] 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*.

[30] 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.

[31] Vladimir I. Levenshtein. Binary codes capable of correcting deletions, insertions, and reversals. *Soviet physics. Doklady*, 10:707–710, 1965.

[32] 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.

[33] 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.

[34] 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.

[35] 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.

[36] Shiqing Luo, Xinyu Hu, and Zhisheng Yan. Holologger: Keystroke inference on mixed reality head mounted displays. In *Proc. of IEEE VR*, 2022.

[37] 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.

[38] 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.

[39] 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.

[40] MediaPipe Hands. Javascript solution api. https://google.github.io/mediapipe/solutions/hands#javascript-solution-api.

[41] 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.

[42] Lawrence Rabiner and Biinghwang Juang. An introduction to hidden markov models. *IEEE Acoustics, Speech, and Signal Processing magazine*, 3(1):4–16, 1986.

[43] 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.

[44] Scott Reed, Honglak Lee, Dragomir Anguelov, Christian Szegedy, Dumitru Erhan, and Andrew Rabinovich. Training deep neural networks on noisy labels with bootstrapping. *arXiv preprint arXiv:1412.6596*, 2014.

[45] Mohd Sabra, Anindya Maiti, and Murtuza Jadliwala. Zoom on the keystrokes: Exploiting video calls for keystroke inference attacks. *CoRR*, abs/2010.12078, 2020.

[46] SeleniumHQ. SeleniumHQ/selenium: A browser automation framework and ecosystem. https://github.com/SeleniumHQ/selenium.

[47] Diksha Shukla, Rajesh Kumar, Abdul Serwadda, and Vir V. Phoha. Beware, your hands reveal your secrets! In *Proc. of ACM CCS*, 2014.

[48] 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.

[49] University of Notre Dame. The frequency of the letters of the alphabet in english. https://www3.nd.edu/~busiforc/handouts/cryptography/letterfrequencies.html.

[50] Andrew Viterbi. Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE transactions on Information Theory*, 13(2):260–269, 1967.

[51] He Wang, Ted Tsung-Te Lai, and Romit Roy Choudhury. MoLe: Motion leaks through smartwatch sensors. In *Proc. of MobiCom*, 2015.

[52] Jiayi Wang et al. RGB2Hands: Real-time tracking of 3D hand interactions from monocular RGB video. *ACM Trans. Graph.*, nov 2020.

[53] Saining Xie, Ross B. Girshick, Piotr Dollár, Zhuowen Tu, and Kaiming He. Aggregated residual transformations for deep neural networks. *CoRR*, abs/1611.05431, 2016.

[54] 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.

[55] 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.

[56] 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.

[57] 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.

[58] 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.

[59] 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.

[60] 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.

[61] 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.

[62] Hongyi Zhang, Moustapha Cissé, Yann N. Dauphin, and David Lopez-Paz. mixup: Beyond empirical risk minimization. *CoRR*, abs/1710.09412, 2017.

[63] 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.

[64] 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.

[65] Li Zhuang, Feng Zhou, and J. Tygar. Keyboard acoustic emanations revisited. In *Proc. of ACM CCS*, 2005.

Appendix

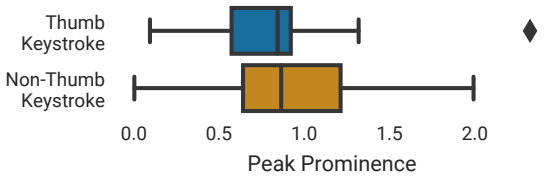

Figure 7: Distribution of negative acceleration's peak prominence for thumb-based and non-thumb keystrokes.

	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 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).

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 emppo 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 8: Examples of final recovered text compared to ground truth.