A Keylogging Inference Attack on Air-Tapping Keyboards in Virtual Environments

Ülkü Meteriz-Yıldıran[*][†] University of Central Florida
Necip Fazıl Yıldıran [‡][§] University of Central Florida
Amro Awad[¶] North Carolina State University
David Mohaisen[‖] University of Central Florida

Abstract

Enabling users to push the physical world's limits, augmented and virtual reality platforms opened a new chapter in perception. Novel immersive experiences resulted in the emergence of new interaction methods for virtual environments, which came with unprecedented security and privacy risks. This paper presents a keylogging inference attack to infer user inputs typed with in-air tapping keyboards. We observe that hands follow specific patterns when typing in the air and exploit this observation to carry out our attack. Starting with three plausible attack scenarios where the adversary obtains the hand trace patterns of the victim, we build a pipeline to reconstruct the user input. Our attack pipeline takes the hand traces of the victim as an input and outputs a set of input inferences ordered from the best to worst. Through various experiments, we showed that our inference attack achieves a pinpoint accuracy ranging from 40% to 87% within at most the top-500 candidate reconstructions. Finally, we discuss countermeasures, while the results presented provide a cautionary tale of the security and privacy risk of the immersive mobile technology.

Index Terms: Security and privacy—Privacy protections; Human-centered computing—Text input; Human-centered computing—Ubiquitous and mobile devices

1 Introduction

After decades of research and development [11, 12, 15, 19, 23], augmented reality (AR) technologies are now available for consumers, offering immersive experiences with a blend of virtual and real-world contents and promising many practical applications [31, 38]. Having got the attention of users via smartphone AR [44, 46], the development of more sophisticated AR technologies, such as head-mounted displays (HMDs), navigation systems [14, 36], and automotive AR windshields [17, 45], is also gaining speed. The applications offered within the recently available AR HMD devices, such as Magic Leap 1 [4] and Microsoft HoloLens [5], show the diverse use cases of these technologies, including simulation, entertainment, training, and assistance. With AR's involvement in response to the needs that emerged from the recent COVID-19 outbreak [3, 16, 22, 34], the benefits of the AR technology are even more evident.

Despite their benefits, new security and privacy risks emerge from the AR/VR's fundamentally new interaction methods. As AR/VR devices get closer to end-users, the community has put efforts into identifying potential risks. HMDs are considered more secure as only the user can see the contents [29]. However, recent work showed they bring risks in other respects [26, 27, 30, 35, 40, 47].

[*]e-mail: meteriz@knights.ucf.edu
[†]Co-first author.
[‡]e-mail: yildiran@knights.ucf.edu
[§]Co-first author.
[¶]e-mail: ajawad@ncsu.edu
[‖]e-mail: mohaisen@ucf.edu

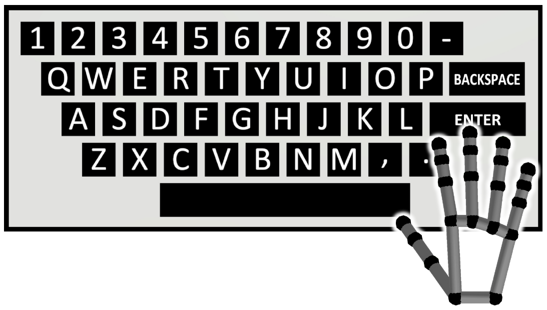


Figure 1: First person view of the AR keyboard and hand model.

The interaction methods in AR/VR and with immersive platforms shift towards more natural methods to offer a more convincing perception [2, 20, 48–52]. These methods include interaction through voice commands, hand gestures, head movements, IMU rings, and remote controllers. For text entry, Microsoft HoloLens uses head movements and hand gestures, Microsoft Hololens 2 uses air-tapping, and Magic Leap 1 uses a remote controller to write on a virtual keyboard with a conventional layout shown only to users.

Acknowledging the risks, this paper presents and evaluates a keylogging inference attack on in-air tapping input entry methods by exploiting the hand trace information. Our attack leverages the fact that hands follow specific patterns when tapping keys to achieve keystroke detection. By localizing the keystrokes in time and space and interpreting the task as a temporal localization problem, we rigorously explore possible keyboard reconstructions from the key tap points and develop heuristics to achieve best to worst ordering of the reconstructions. Our method avoids any assumption on the positioning and size of the keyboard to account for users' freedom in placement, configuration, and scale of virtual objects, which is an essential feature of AR/VR applications. Our experiments demonstrate that we can recover the text typed by a victim user with up to 87% pinpoint accuracy. Additionally, the results remain above 40% for text length varying from 5 to 250, which is alarming.

Contributions. The research community envisions that the AR/VR technology will invade many new domains and will be used by a large number of users ubiquitously [28, 39], which makes understanding the technology's potential vulnerabilities both timely and critical. To this end, our key contributions are as follows. **(1)** We present the first inference attack on in-air tapping keyboards for AR/VR devices. Our attack is based on the victim's hand traces and does not require any user profiling, making it applicable to various scenarios and adversarial capabilities. Our attack avoids strong assumptions on the positioning and scaling of the keyboard to account for users' flexibility in configuring them in virtual environments. **(2)** We present a detailed step-by-step and end-to-end evaluation of a five-step attack pipeline. We measure the performance of our method in terms of various metrics from the related literature. We discuss how the performance of each step affects the end results and demonstrate the performance of the disjoint steps to show their applicability to keylogging inference attacks in various contexts. **(3)** By observing that our attack and the use scenarios assume a fixed position keyboard in a single session, we propose and analyze a defense by dynamically positioning the keyboard in space.

Organization. We describe the system and threat models in § 2, followed by the presentation of our five-step pipeline in § 3. We discuss the performance evaluation, results, and defenses in § 4. We review the related work in § 5 and conclude our work in § 7.

2 System and Threat Models

2.1 System Model

For executing our attack, we developed an in-air tapping keyboard–*AR keyboard*. Figure 1 shows the first person view of the AR keyboard and the hand models. Although the keyboard design and our study also apply to VR, we mainly address AR environments.

AR Keyboard Design. In our keyboard design, the user attaches the AR keyboard to some arbitrary location in the virtual environment, and the AR HMD tracks the user's hands. The AR HMD detects the keystrokes by checking if a fingertip collides with a key in space. The keyboard layout is US and it can be sized by the user freely.

Session. We define a **session** as the process of typing a text on the AR keyboard. We assume the keyboard has a fixed spatial configuration during a session, although it may change between sessions.

Tracking. Hand tracking sensors provide spatial information (i.e., positions, angles) of hands and fingers. For users to type on the AR keyboard and for the adversary to carry out the attack, hand tracking data is essential. For users, hand tracking capabilities are built-in in AR HMDs [2, 43]. For the adversary, the hand tracking data comes from one of a few possible mediums, as explained in § 2.2. Each of those mediums provides the following information: the tip positions of all fingers, the pointing directions of all fingertips, the position of the palm center, and the palm normal. We refer to this information as the **low-level hand tracking data**. We expect that our approach will work with minor modifications even when the low-level hand tracking data is not available precisely in the same format, but as long as the spatial information of the hands is available.

2.2 Threat Model

We envision a concrete threat model that considers a victim using the AR keyboard in AR HMD, and an adversary attempting to infer what the victim types by exploiting the hand traces. To achieve this goal, the adversary gains access to the victim's hand traces through one of a few possible mediums, each of which is covered in an attack scenario. Other than differences in the method used for obtaining the hand tracking data, the three scenarios, depicted in Figure 2, share the same procedure from the adversary's standpoint. Once the adversary obtains the hand traces, she will be able to perform the inference attack offline. With the innovations in AR domain, the industry leaders launched various types of AR HMDs [4–7, 9] for various stakeholders with countless use cases. Such speed in development and deployment cycle indicates that AR HMD technologies are expected to be an asset for daylong use [28, 39]. Given the consumer demand in AR technologies and the rapid advancements, the ubiquitous usage of AR HMDs in near future is plausible.

Scenario 1. In this scenario, the adversary plants a hand tracker device near the victim in a public space, e.g., a coffee shop, and records his hand movements as the user types in the AR keyboard. The adversary might use the Leap Motion Controller [8] as the hand tracker—being small (8x3x1.1cm) allow hiding in a stealthy attack. Although the off-the-shelf version of the controller requires cable connection, wireless setups are available [1] and could be utilized to reduce the cable burden for a stealthier attack instance.

Scenario 2. In this scenario, the adversary wears an AR HMD and sits near the victim, possibly in a public space, to record the hand movements of the victim using the hand tracking feature of the AR HMD. Due to the expected ubiquitous usage of AR HMD, it is challenging to identify the adversary. The adversary needs to be close to the victim (approx. 1.5m), where the data collection is much easier to carry out in context, e.g., two acquaintances in the same space, where one of them might be curious what the other is typing.

Scenario 3. In this scenario, the adversary infects the AR HMD of the victim with a malware, enabling the adversary to obtain the data from the built-in hand tracking sensor through the AR HMD's API. One advantage of this scenario is that there is no assumption on where the victim is located nor the adversary's proximity to the victim. As such, the attack could be conducted remotely, at any time, from anywhere, without needing to deploy any additional device near the victim. We should note that obtaining the sensor readings is a more straightforward attack than reading the processed outputs of the keyboard application. Malware can obtain the sensor readings using the device API stealthily; however, reading the keyboard's processed output requires more sophisticated malware. For example, assuming the gyroscope readings are available to the attacker does not mean that the attacker can directly read the keyboard's output.

We should note that the adversary does not know the position of the AR keyboard in the victim's virtual environment since users can change the position however they like. Therefore, for all of the adversarial scenarios above, the sensor readings cannot enable the adversary to directly obtain the keyboard output by simply forwarding the sensor readings to the keyboard application. In all scenarios, a calibration between the coordinate system of the victim's virtual environment and the coordinate system of the virtual environment model of the adversary is required, and this requirement is one of the main technical challenges we overcome in this study.

3 Technical Details and Methods

To set out, In figure 3 we outline the technical details of our five-step attack pipeline. Keylogging inference attacks are defined as a two-stage process: keystroke detection and key identification [37]. The first two steps of our pipeline are for keystroke detection while the subsequent three steps are for key identification.

3.1 Deep Key Tap Localization

To initiate our attack, we start by localizing the key taps on the low-level hand tracking data stream using the following observation.

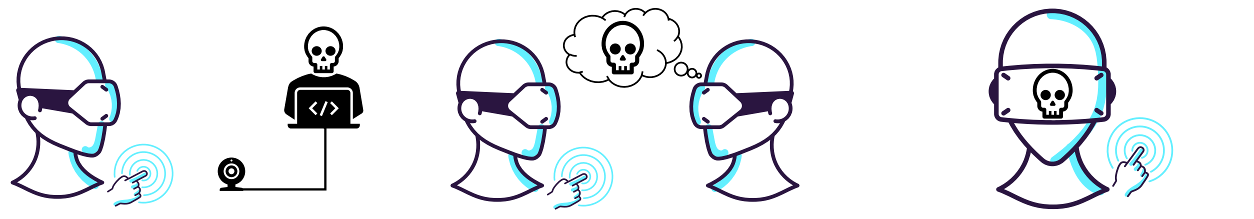
Observation 1. *The hands follow a certain pattern while writing on the AR keyboard. This, in turn, makes the key tap gestures distinguishable from all other hand gestures.*

To exploit Observation 1, we utilize a Convolutional Neural Network (CNN) to localize key taps in time windows where key tap-specific patterns are observed. However, the time-domain low-level hand tracking signal changes as the tracking sensor is placed differently. Therefore, we pre-process the data to resolve this dependency and ensure the attack repeatability in different spatial configurations.

Pre-processing. In this phase, we obtain three features from the low-level hand tracking data: 1) the tip position of each finger w.r.t. the palm center of the hand (f_p) by subtracting the position of the palm center from the position of the fingertips, 2) the pointing direction of each finger w.r.t. to the normal vector of the palm (f_d) by subtracting the normal vector of the palm from the direction vectors of the fingers, and 3) the velocity of each fingertip (f_v) by calculating the replacement of the fingertips from f_p. We call them position, direction, and velocity sub-signals, respectively.

We denote hand features as $X = \{x_t\}_{t=1}^{T}$ where the t-th frame is $x_t = [f_p\ f_d\ f_v]^T$. A segment $s(t_{start}, t_{end})$ is a slice that includes the frames between x_{start} and x_{end}.

Network Architecture. Figure 4 illustrates the multi-head CNN architecture. Each head is designed to fetch the independent temporal features in the time series-like data. The kernel size of the convolutions cover 3 frames in time and perform convolutions with

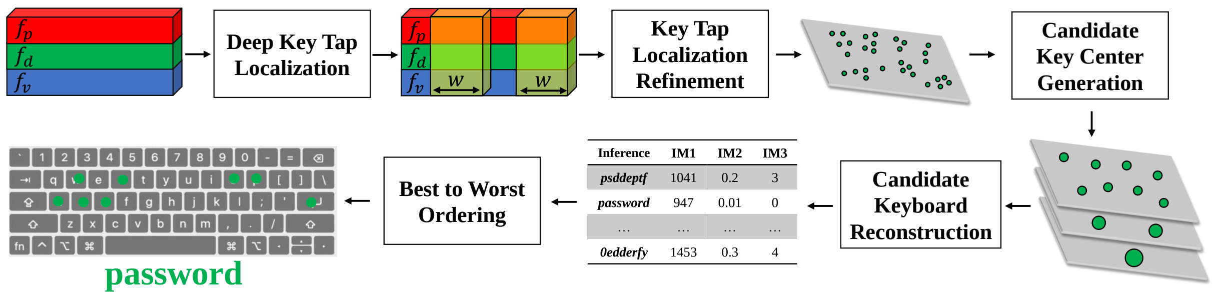


(a) Scenario 1: The adversary plants a hand tracker device and records the victim's hand movements.

(b) Scenario 2: The adversary sits near the victim and records his hand movements using AR HMD.

(c) Scenario 3: The adversary infects the victim's AR HMD and remotely obtains the tracking data.

Figure 2: Illustrations of the attack scenarios with varying adversarial capabilities.


f_p
f_d
f_v
Deep Key Tap Localization
w
w
Key Tap Localization Refinement
Candidate Key Center Generation
Candidate Keyboard Reconstruction
Best to Worst Ordering
password

Inference	IM1	IM2	IM3
psddeptf	1041	0.2	3
password	947	0.01	0
…	…	…	…
0edderfy	1453	0.3	4

Figure 3: The five-step pipeline for the attack, which takes hand tracking data as input and outputs inferences ordered from best to worst. The first two steps are for keystroke detection stage while the subsequent three are for key identification stage.

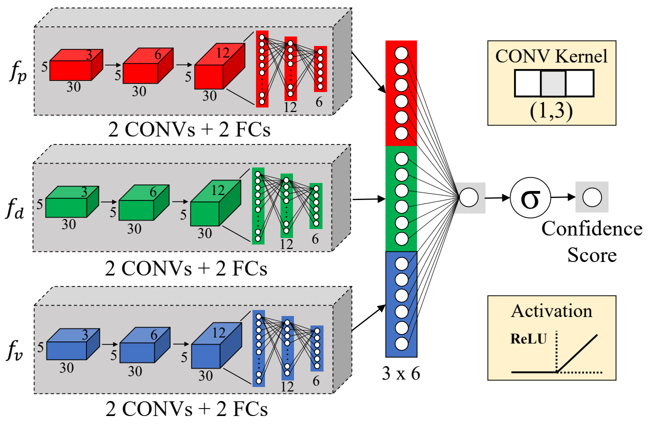


Figure 4: The multi-head CNN architecture used for key tap localization. The input is a segment from the pre-processed low-level hand tracking data, while the output is the confidence score for the segment belonging to a key tap.

stride 1. The rationale of the multi-head design is that data included in the segments are initially independent from each other. Concatenating them and passing them to a single CNN where the stride is set to 1 would cause irrelevant convolutions around the edges. For example, it is redundant to convolve the position of the pinky finger with the direction of the thumb. In other words, convolving position and direction is not desired in the early stages. However, after the convolutional layers and a fully-connected network in the heads, we get a higher-level representation of the data segments, which are now time-independent. Once we extract the time dependency, we concatenate the features and get the confidence scores with a final fully-connected network and a sigmoid function. We interpret each key tap as a 30 frame action, or simply 375 ms, given that the sampling rate of the hand tracker is 80 fps. Therefore, each head takes a sub-segment with the shape of height 5 (fingers), width 30 (frames), and depth 3 (dimensions) as input.

Localization. For localization, we slide a fixed-length (30) window on the low-level data stream to generate segments. We then obtain the confidence score P_c of each segment using the CNN component. We filter out the redundant detections by applying *non-max suppression* to ensure that the localized windows are disjoint and only the most confident windows are kept. Then, we eliminate the segments with P_c less than the classification threshold $\tau_c = 0.5$, with the rationale explained in the evaluations. Finally, each segment is associated with a key tap point by fetching the position of the index fingertip at the midpoint of the window.

3.2 Key Tap Localization Refinement

Since we estimate the *key tap points* in the deep key tap localization stage, they might be imprecise. In this stage, we refine the points in four steps based on a geometric observation on the keyboard.

Observation 2. *The key tap points lay on a plane in 3D.*

Step 1: Keyboard Plane Estimation. As the AR keyboard lays on a plane in 3D, the key taps should also lay on the same plane. Therefore, we first estimate a keyboard plane from the obtained key tap points. A plane in 3D is defined by a normal vector perpendicular to the plane, a point on the plane, and a scalar. To find a fitting plane to the estimated 3D key tap points, we use a regression model that minimizes a linear least square error between the points and plane.

Step 2: Reducing the False Positives. We follow the trace of the fingertip for each key tap window and check if it crosses the estimated plane. Otherwise, we eliminate it since the finger does not touch the keyboard plane.

Step 3: Refining the Key Tap Points. To increase the spatial precision of the key tap points, we use the intersection point where the fingertip crosses the estimated keyboard plane instead of using the midpoint of the key tap window as before.

Step 4: Dimension Reduction. Since all the key tap points lay on the same plane, we can easily and feasibly represent them as 2D points without any loss of information. To achieve this task, we first create an orthonormal basis containing the normal of the plane using the Gram-Schmidt Orthogonalization. Then, we change the basis

Algorithm 1 Construction of center groups

```
1: $G$: Cluster groups with varying number of clusters
2: $C$: Current set of weighted centroids
3: $K$: Key tap points
4: procedure BuildCenterGroups($K$)
5:     initialize:
6:         $G \leftarrow \emptyset$
7:         $C \leftarrow \{(\mathbf{v}, w) \in \mathbb{R}^2 \times \mathbb{R} \mid \mathbf{v} \in K, w = 1\}$
8:     while size($C$) $> 2$ do
9:         $\mathbf{c}_1, \mathbf{c}_2 \leftarrow$ MinimumDistancePair($C$)
10:        $\mathbf{c}_{merged} \leftarrow$ MergeCenters($\mathbf{c}_1, \mathbf{c}_2$)
11:        $C \leftarrow (C \cup \{\mathbf{c}_{merged}\}) \setminus \{\mathbf{c}_1, \mathbf{c}_2\}$
12:        $G \leftarrow G \cup \{C\}$
13:    return $G$
14: procedure MinimumDistancePair($C$)
15:    return ($\mathbf{c}_1, \mathbf{c}_2 \mid \mathbf{c}_1, \mathbf{c}_2 \in C$, EucDist($\mathbf{c}_1, \mathbf{c}_2$) is min)
16: procedure MergeCenters($\mathbf{c}_1, \mathbf{c}_2$)
17:    $(\mathbf{v}_1, w_1), (\mathbf{v}_2, w_2) \leftarrow \mathbf{c}_1, \mathbf{c}_2$
18:    $\mathbf{v}_{merged} \leftarrow$ WeightedVectorAvg($\mathbf{c}_1, \mathbf{c}_2$)
19:    $w_{merged} \leftarrow w_1 + w_2$
20:    return $(\mathbf{v}_{merged}, w_{merged})$
```

for the key tap points. Finally, we obtain 2D key tap points K by omitting the component in the direction of the normal vector.

3.3 Candidate Key Center Generation

In this step, we deduce candidates for the key centers of the AR keyboard from the refined key tap points. We set forth the following observation about the relationship between the key centers and the key tap points, which we exploit for our deduction.

Observation 3. *When each key center in the AR keyboard is considered as the cluster centroid of that particular key, each key tap point will belong to the cluster of the tapped key.*

Based on Observation 3, we hypothesize that a proper clustering of the key tap points eventuates partitioning them into clusters of unique keys. For proper clustering, the number of clusters should be the same as the number of unique keys pressed in a session. However, not all unique keys are necessarily used during a session. Thus, the attacker is unaware of the number of clusters required, which is a key challenge in this step. To address this issue, we generate cluster groups with a varying number of clusters through the agglomerative hierarchical clustering to account for varying numbers of keys.

We denote a cluster group as $C = \{\mathbf{c}_m\}_{m=1}^{M}$ where $\mathbf{c}_m$ is the 2D vector representing of the centroid of the m-th cluster in the group C. The output of this step is a set of cluster groups, G, where each has a different cluster count. The set of cluster groups is denoted by $G = \{C_n\}_{n=1}^{N}$, where C_n is the n-th clustering.

Algorithm 1 shows the procedure we use to compose the cluster groups G. Initially, all key tap points are treated as singleton clusters with uniform weights as in line 1.7. Then, the cluster group C is iteratively updated (line 1.11) by merging the closest clusters considering the weights. Each update to the cluster group C results in adding a cluster group to G (line 1.12) with a different cluster count. The Euclidean distance is then used to find the closest cluster pair, and merging is done through weighted vector average.

3.4 Candidate Keyboard Reconstruction

In this step, we output the possible keyboard reconstructions by overcoming two challenges.

Challenge 1: Unknown Coordinate Systems. Since the hand tracker and the adversary's AR keyboard model have different coordinate systems, the same geometric constructs (e.g., points, vectors) in a shared environment (i.e., real-world) are expressed differently by those two parties. Moreover, the user's positioning and scaling of the AR keyboard differently in the virtual environment introduce yet another discrepancy. Considering these series of linear transformations between the coordinate systems, we realize a similarity transform T which represents the mapping between the coordinate systems. Using two corresponding point pairs from each space, T could be computed as shown in Algorithm 2. We refer to T( ) as the similarity transformation function that applies T to its input.

Algorithm 2 The algorithm to compute similarity transformation T using two pairs of corresponding points in 2D. It can be used to map any point $\mathbf{a}$ in space A to its correspondent $\mathbf{b}$ in space B.

```
1: procedure ComputeTransformation($\mathbf{a}_1, \mathbf{a}_2, \mathbf{b}_1, \mathbf{b}_2$)
2:     $\mathbf{t} \leftarrow \mathbf{b}_1 - \mathbf{a}_1$                                  ▷ translation vector
3:     $s \leftarrow \frac{|\overrightarrow{\mathbf{a}_1\mathbf{a}_2}|}{|\overrightarrow{\mathbf{b}_1\mathbf{b}_2}|}$                      ▷ uniform scaling scalar
4:     $\alpha \leftarrow \cos^{-1} \frac{\overrightarrow{\mathbf{a}_1\mathbf{a}_2} \cdot \overrightarrow{\mathbf{b}_1\mathbf{b}_2}}{|\overrightarrow{\mathbf{a}_1\mathbf{a}_2}| \cdot |\overrightarrow{\mathbf{b}_1\mathbf{b}_2}|}$         ▷ rotation angle
5:     return FormMatrixT($\mathbf{t}, s, \alpha$)                       ▷ see [41]
```

Algorithm 3 Reconstruction of candidate keyboards

```
1: $G$: Cluster groups with varying number of clusters
2: $K$: Key tap points
3: $J$: 2D centroids of the keys in the AR keyboard model
4: $L$: The candidate keyboard reconstructions
5: procedure ComputeCandidRecons($G$)
6:   initialize: $L \leftarrow \emptyset$
7:   for each $C$ in $G$ do
8:     for each permutations of centers $\mathbf{c}_1, \mathbf{c}_2$ from $C$ do
9:       for each combinations of centroids $\mathbf{j}_1, \mathbf{j}_2$ from $J$ do
10:        $T \leftarrow$ ComputeTransformation($\mathbf{c}_1, \mathbf{c}_2, \mathbf{j}_1, \mathbf{j}_2$)
11:        $K' \leftarrow$ T($K$)
12:        $L \leftarrow$ InferKeys($K'$) $\cup L$
13:   return $L$
```

Challenge 2: Unknown Corresponding Points. Having T, the key tap points K can be transformed into the space of the AR keyboard model as $K' = \mathrm{T}(K)$. Then, each key tap position k' in K' could be associated with a key by checking which key's area k' falls onto. This results in recovering the keys tapped. However, the adversary lacks two corresponding points from both spaces, failing to directly obtain the correct transformation. To address this challenge, we use the key centers from the previous step. As there is no prior knowledge on which cluster corresponds to which key, we account for the possibility of each cluster c belonging to any unique key j for each cluster group C in G. We call a pairing of cluster center $\mathbf{c}$ with a key $\mathbf{j}$ as **center – key pair**. Then, we obtain candidate keyboard reconstructions by computing T using every possible center – key pairs. We summarize the reconstruction of candidates in Algorithm 3.

3.5 Best to Worst Ordering of Reconstructions

The previous step produces numerous keyboard reconstructions, due to its exhaustive process. However, many of them are inaccurate due to the incorrect center – key pairings. Although the search space is significantly reduced compared to having random guesses, it is still infeasible for the adversary to consider all reconstructions. We observe that accurate reconstructions typically have specific characteristics. In this step, we measure these characteristics using Inference Measurements (IM) to order the keyboard reconstructions from best to worst, drastically decreasing the number of reconstructions to consider.

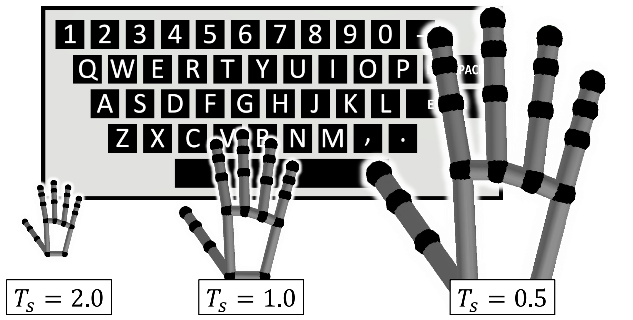


Figure 5: The demonstration of the maximum and the minimum scaling factors (**IM$_1$**) used to filter the candidate keyboard reconstructions. For concise visualization, the effect of T_s is shown by inversely scaling the hand model rather than scaling the keyboard itself.

IM$_1$: Scaling Factor. Users are free to scale the AR objects in the virtual environment controlled by T_s. Making the keyboard too small or too large negatively affects usability. As shown in Figure 5, a smaller keyboard ($T_s = 0.5$) makes it difficult to pick the correct key. In comparison, a larger keyboard ($T_s = 2.0$) increases the distance the hands have to travel. Moreover, with a too large keyboard, the keyboard may not fit into the frame of the AR HMD. Therefore, we define limits for T_s to eliminate infeasible reconstructions.

IM$_2$: Outlier Ratio. All key tap points should optimally fall into the area of some key in the AR keyboard. However, we observe outlier key tap points in improper reconstructions. To measure such anomaly, we define **IM$_2$** and compute it as the ratio of the number of outliers to all key tap points. Although the optimal value for **IM$_2$** is zero, we expect a smaller number of outliers.

IM$_3$: The Number of Clusters – Unique Keys. The correct clustering of key tap points should divide the key tap points into clusters of unique keys. For a correct keyboard reconstruction, we expect the number of clusters to be optimally equal to the number of unique keys found after the transformation. For counter cases, we measure the anomaly using **IM$_3$**, which is the difference between the number of clusters and the found unique keys. The optimal value for **IM$_3$** is zero, yet some cases may result in values not equal but close to zero for accurate inferences.

Upon computing IMs, we eliminate the reconstructions by enforcing a scaling factor limit (**IM$_1$**). However, we do not directly enforce the optimal values for **IM$_2$** and **IM$_3$**. To capture and estimate the joint role of **IM$_2$** and **IM$_3$** on the reconstruction's correctness, we utilize a linear regression model. The model takes **IM$_2$** and **IM$_3$** as input, and outputs the associated correctness score.

We train the regression model with the correctness scores measured in terms of the normalized Levenshtein similarity between the estimated and the ground truth strings. Levenshtein similarity measures the performance of both the keystroke detection and key identification stages simultaneously [37]. After we train the regression model, we forward **IM$_2$** and **IM$_3$** of each candidate reconstructions to it. Then, we sort the reconstructions in a decreasing order w.r.t. their correctness score. These sorted reconstructions constitute the final output of our pipeline, i.e., the best to worst ordering of the keyboard reconstructions. Note that once the correct keyboard reconstruction is found, it is trivial to trace any input on the keyboard, including the special characters.

4 Evaluation

In our experiments, we evaluated each step of the pipeline and end-to-end. With the evaluations of the individual steps, we intend to: (1) reason about the validity of our assumptions which led to construct each step, (2) explore how each step contributes to the overall pipeline, and (3) explore the limitations of each step and discuss their effects on the overall approach. The experiments involved five participants, including one female and four males with ages ranging from 23 to 37. The Institutional Review Board (IRB) at our institution approved the data collection process.

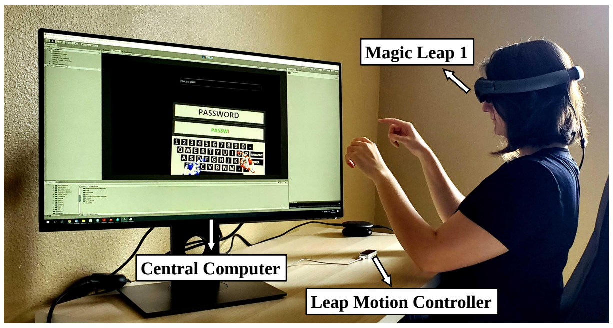


Figure 6: The setup used to carry out the experiments.

4.1 Experimental Setup

All threat models exploit low-level hand tracking data. Therefore, we simulate and evaluate them with the same setup irrespective of the data collection medium. For our experiments, we use Magic Leap 1 (ML1) as the AR HMD, and Leap Motion Controller (LMC) as the hand tracker, as shown in Figure 6. LMC can track hands within 60 cm in a 120×150° field of view, and ML1 detects hands within 80 cm distance.

Metrics. We measure the accuracy of a text inference in terms of the normalized Levenshtein similarity. We also use pinpoint, *h*-hop, and top-*k* accuracy [42]. The *h*-hop accuracy is computed by considering the predicted keys that can be reached by taking *h* hops from the actual key as true. The *h*-hop metric shows how seemingly inaccurate inferences still cause privacy leaks. The pinpoint accuracy is 0-hop accuracy. Top-*k* accuracy is the maximum accuracy within the first *k* inferences in the best to worst ordering. Top-*k* is essential to evaluate our method since we output a set of inferences.

4.2 Keystroke Detection

For this set of evaluations, we collected the **P108** dataset from two users while they were typing 54 random pangrams, mainly used to simulate the data recorded by the adversary for training purposes. The data is then split into train-validation (96 pangrams) and test (12 pangrams) splits. To create the ground truth, we fetched the key tap segments from each sample and labeled them as 1. Similarly, we fetched disjoint background segments from the remaining background data and labeled each of them as 0 for convenience.

Deep Key Tap Localization. To train the multi-head CNN, we use Adam optimizer [24] with a learning rate of 0.001, L2 regularization with a penalty of 0.001, and the weighted binary cross-entropy loss function. The hyperparameters are determined by 5-fold cross-validation. Then, we perform predictions on the test split. The prediction procedure takes a set of input segments $S_{\text{test}} = \{s_n\}_{n=1}^N$ and returns the key tap segments with confidence scores greater than a classification threshold: $P = \{(s_m, \alpha_m)\}_{m=1}^M$. Given each ground truth key tap segment gt_l in $GT = \{gt_l\}_{l=1}^L$, we associate each key tap segment s_m in P with a ground truth segment gt_l, which is closest to s_m in time. For each association, we measure the intersection-over-union (IoU). For two segments, $S_1(t_1,t_2)$ and $S_2(t_3,t_4)$, where t_1 and t_3 are the start time, t_2 and t_4 are finish time, and $t_3 > t_1$, IoU between S_1 and S_2 is defined as $min(0, t_2 - t_3)/(t_4 - t_1)$, which takes values in $[0,1]$. IoU becomes 1 if the segments fully overlap.

As a result, we obtain a set of associations of segments and their IoU values: $A = \{(s_m, gt_l, \text{IoU})\}$. We eliminate the associations with IoU $< \tau_{\text{IoU}}$, where τ_{IoU} is the IoU threshold. This helps us evaluate the model over different IoU thresholds to reason about the precision of the model in temporal localization. Following the

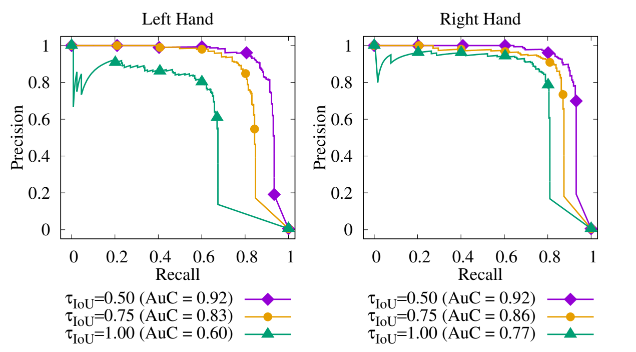


Figure 7: The precision-recall curve of the multi-head CNN in *Deep Key Tap Localization*. Different networks are trained for each hand, and both of them perform similarly well.

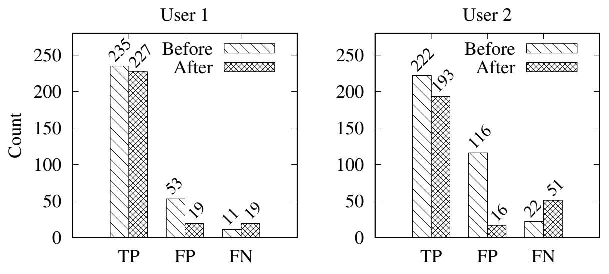


Figure 8: The effect of the *Key Tap Localization Refinement* on the number of true positive (TP), false positive (FP), and false negative (FN) samples for different users.

standard evaluation methods in temporal localization problems, we interpret the segments in A as true positives, $P \setminus A$ as false positives, $GT \setminus A$ as false negatives, and the remaining in S_{test} as true negatives.

Based on our preliminary observations during our experiments, we set the classification threshold as $\tau_c = 0.5$, where τ_c controls the trade-off between true positives and true negatives. More precisely, $\tau_c = 0.5$ favors a greater number of the true positive samples while keeping the false positives less than the true positives.

Finally, the Precision-Recall (PR) curves for left and right hand and varying τ_{IoU} values are shown in Figure 7.

Key Tap Localization Refinement. In this step, we eliminate a significant number of false positives, which is not possible by just tweaking the classification threshold. To assess the performance of this step, we performed *Key Tap Localization Refinement* on the output of the previous step. As shown in Figure 8, the refinement procedure significantly decreases the number of false positives (by 64.1% and 86.2%, respectively) while slightly affecting the true positives (decreased by 3.4% and 13.1%, respectively).

4.3 Key Identification

For this set of evaluations, we collected data from three users (other than P108's users) while they were writing an e-mail randomly selected from the Enron dataset [25]. For length uniformity, we use the first 250 characters for each recording. We denote this data as **E-mail** data. Additionally, we collected **R5**, **R10**, and **R15**, while the users were writing random character sequences with varying lengths of 5, 10, and 15. We collected 15 different sequences for each length and the reported results are the average of the 15 sequences. We aim to simulate the long-text scenarios, e.g., writing an e-mail, with the **E-mail** dataset while using **R5**, **R10**, and **R15** for simulating the entry of short-text scenarios, e.g., login credentials.

To evaluate the key identification stage only, we use User-1's data and directly use the ground truth key tap points. We first perform *Candidate Key Center Generation*, then *Candidate Keyboard Reconstructions*. This results in 889,200, 9,750, 52,000, and 126,750 keyboard reconstructions for e-mail and each sequence in **R5**, **R10**, and **R15**, respectively. The difference in the number of keyboard reconstructions is due to the number of key taps, and it is upper-bounded by 889,200 reconstructions due to the upper bound on the number of centers. Among those, we are able to achieve a maximum pinpoint (h_0) accuracy of **97**% for **E-mail**, **99**%, **100**%, and **99**% for **R5**, **R10**, and **R15** sets, respectively, and all reach **100**% of h_1 and h_2 accuracy. Although not all reconstructions have the same accuracy, the results show that the key identification successfully includes accurate reconstructions. The evaluation results of **R5**, **R10**, and **R15** also demonstrate that our method works with a limited number of samples, as low as 5, to create the correct keyboard reconstruction.

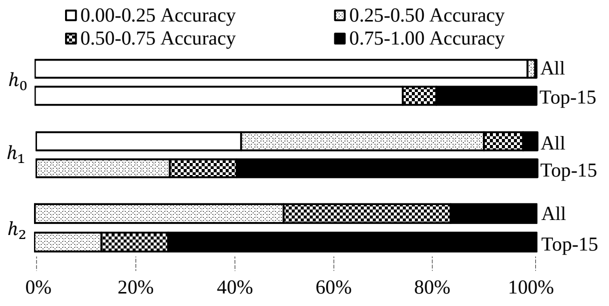


Figure 9: The accuracy distribution of the E-mail keyboard reconstructions before and after best to worst ordering. "All" corresponds to reconstructions after $\mathbf{IM_1}$ filtering.

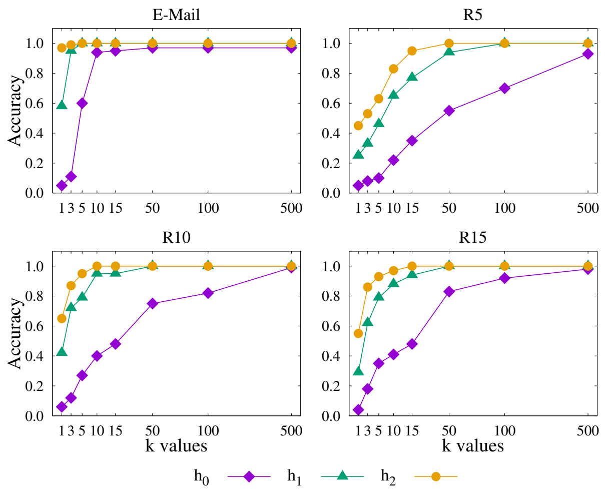


Figure 10: The top-k h_0, h_1, and h_2 accuracy for varying k considerations for each data set obtained by utilizing the key identification steps on the ground truth key tap points.

Upon obtaining the reconstructions, we continue with the best to worst ordering by first filtering the reconstructions based on the minimum and the maximum scaling factors ($\mathbf{IM_1}$) and consider the pinpoint (h_0) accuracy in evaluating its effect. The filter extracts over 40% of the reconstructions with an accuracy that ranges between 0% and 25%, and keeps all reconstructions with accuracy that is higher than 50%. We then ordered the rest thorough $\mathbf{IM_2}$ and $\mathbf{IM_3}$. We fit a linear regression model using the **P108** data, and obtain the best to worst ordering for the reconstructions belonging to **E-mail**, **R5**, **R10**, and **R15**. Figure 9 shows the distribution of all reconstructions (after $\mathbf{IM_1}$ filtering) and the top-15 reconstructions for **E-mail** data on different accuracy quarters.

The results highlight that ordering indeed uplifts the better reconstructions to the top of the list and significantly increases the probability of finding an accurate reconstruction even when randomly chosen from the first 15 elements of the list. Figure 10

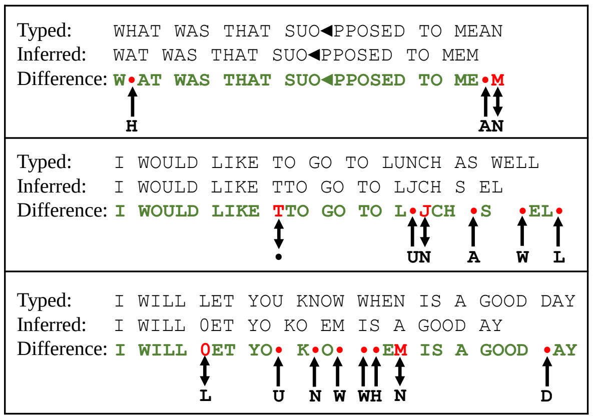

Typed: WHAT WAS THAT SUO◀PPOSED TO MEAN
Inferred: WAT WAS THAT SUO◀PPOSED TO MEM
Difference: W•AT WAS THAT SUO◀PPOSED TO ME•M
H
AN
Typed: I WOULD LIKE TO GO TO LUNCH AS WELL
Inferred: I WOULD LIKE TTO GO TO LJCH S EL
Difference: I WOULD LIKE TTO GO TO L•JCH •S •EL•
•
UN
A
W
L
Typed: I WILL LET YOU KNOW WHEN IS A GOOD DAY
Inferred: I WILL 0ET YO KO EM IS A GOOD AY
Difference: I WILL 0ET YO• K•O• ••EM IS A GOOD •AY
L
U
N
W
WH
N
D

Figure 11: The 11th inference found in the ordered list which is obtained utilizing the end-to-end pipeline on the E-mail. The spaces are inserted for clearer visualization.

demonstrates the top-k accuracy for varying k using h_0, h_1, and h_2 metrics. Over 95% h_2 accurate reconstructions are listed within top-15 of the ordered list. The reconstructions with over 92% h_0 accuracy are listed within the top-500. As the number of key tap points scattered over the AR keyboard increases, the shifted reconstructions are more easily detected with a higher $\mathbf{IM_2}$. For example, two centers representing the keys A and D are matched with the AR keyboard model's keys Q and E, causing the keys to be recovered as the keys above them, causing a *shift*. Hence, a higher h_0 accuracy is reached in the earlier top-k considerations when the number of key tap points is larger. Yet, having h_0 inaccurate results does not necessarily amount to entirely irrelevant results. Obtaining much higher h_1 and h_2 for such low h_0 values shows that the low pinpoint accuracy is compensated with higher h_1 and h_2 accuracy due to the existence of shifted reconstructions.

4.4 End-to-End Pipeline

In this section, we present end-to-end pipeline results for all datasets and users. Using the **P108** data, we first train the two models, the multi-head CNN for *Deep Key Tap Localization* and the linear regression for *Best to Worst Ordering*. Next, we feed the **E-mail**, **R5**, **R10**, and **R15** datasets to the pipeline. We use the same hyperparameters determined in the evaluation of the individual steps.

Table 1 shows the values of the h_0, h_1, and h_2 accuracy obtained for each dataset and each user. These values are obtained considering the top-k with select values of k and the average accuracy obtained by randomly guessing each character, i.e., brute-force attack.

Overall, the results show that our model can successfully infer the text with a high accuracy. For edge cases where the pinpoint (h_0) accuracy is close to that of a random guess, the h_1 and h_2 accuracy results show that the inaccurate key estimations of our method appear close to the actual key, decreasing the search space for the adversary. For example, even though the pinpoint (h_0) accuracy of the first guess (Top-1) of our model for **R15** data of User-3 is the same as that of random guess, i.e., 0.07, our method performs much better in terms of h_1 (0.38 vs. 0.18) and h_2 (0.57 vs. 0.35) accuracy.

The maximum h_0 achieved over all reconstructions for **E-mail**, **R5**, **R10**, and **R15** is 68%, 76%, 87%, and 62%, respectively. These reconstructions are populated within the top-500 results. Comparing the results with those in § 4.3 shows a clear evidence of the propagation of errors in the keystroke detection to the end results, which prevents the pipeline from achieving 100% accuracy.

Finally, Figure 11 shows the first three sentences of User-1's e-mail data and the inference of those sentences found at 11th place in the ordered reconstructions list. In these inferred sentences, there are 12 missing, 1 extra, and 3 incorrectly identified letters. The missing letters are due to the false negatives in the keystroke detection and require spotting and guessing for correction. The extra letter is due to the false positives in the keystroke detection and needs to be spotted to fix. Finally, we can easily correct the incorrectly identified letters by replacing them with the close neighbor keys.

4.5 Defenses

The findings we have presented are both promising, from a learning standpoint, and alarming from a security standpoint. To counter the attack, in the following we present various defense mechanisms.

One simple yet highly effective defense to our attack would be by randomizing the keys at each keyboard usage or after every key tap [33]. However, this would affect usability, specifically for long text inputs [32]. Another simple defense would be by invalidating the assumption that *the keyboard lays on a single plane.* By introducing a keyboard design represented by N planes, e.g., a curved keyboard, we could becloud the attack process for the adversary. Another defense is by invalidating the adversary's assumption that the keyboard position remains fixed throughout a session by altering the position of the keyboard after each keystroke, i.e., by dynamic positioning. We describe the implementation and conduct a security analysis to demonstrate the efficacy of this defense.

Dynamic Positioning. A translation vector $\vec{\mathbf{t}}$ is represented with two scalar parameters: its direction angle α and length d. After each keystroke, we use a translation vector $\vec{\mathbf{t}}$ by choosing α and d randomly from the ranges $[0,2\pi]$ and $[r,2r]$, respectively, where r is the diagonal length of a key. The motivation for upper-bounding d by $2r$ is to ensure that the usability is not significantly affected, where the user would quickly locate the key position by taking the former keyboard position as a reference. With the lower bound of r, we guarantee that the keys will always change position by at least one hop between two keystrokes. Finally, randomly choosing d within the $[r,2r]$ range, the adversary is substantially challenged by the fact that any key's position on the keyboard may correspond to any key within the 2-hop neighborhood.

Analysis. For a security analysis of this defense, we consider a few assumptions to make the analysis tractable. We assume that: 1) the adversary achieves keystroke detection with 100% accuracy with perfect spatial localization, 2) the user always taps a key from the key's center point, 3) for each displacement, the translation vector $\vec{\mathbf{t}}$ is chosen to displace the keyboard such that the centers of the alphanumerical keys are moved to exactly one of the keys' centers within their 2-hop neighborhood. These assumptions are unrealistic and only put the adversary in a more favorable position while enabling us to sketch a lower bound on the security analysis. We emphasize that what we do next is a lower bound security analysis, and the exact security of this defense is more robust in reality.

When the initial position of the keyboard is known, to guess the key for the next keystroke, the adversary can exploit the fact that it can only be to one of the keys within the 2-hop neighborhood of the key formerly present at that location. Therefore, for each key k, the probability of successfully recovering k after a displacement is equal to the ratio of the number of positions where k can appear to the number of distinct keys that can appear at these positions. We show the probabilities for each key in Figure 12, from which we derive the average probability as $P_{\text{next}} = 0.25$.

Since the adversary does not know the initial keyboard position, we consider the probability of successfully identifying the first key as $P_{\text{first}} = \frac{1}{36} = 0.028$, where 36 is the number of the alphanumerical keys. Using this analysis, the adversary's success probability to guess a text of length l is $P_l = P_{\text{first}} \times (P_{\text{next}})^{l-1} = 0.028 \times (0.25)^{l-1}$. For example, for texts of length 4 and 8, $P_{l=4} = 4.4 \times 10^{-4}$ and $P_{l=8} = 1.7 \times 10^{-6}$, which shows the effect of the countermeasure for practical scenarios such as inputting passwords.

We anticipate that the dynamically-positioned keyboard will be more usable, especially compared to the randomized keyboard, since

Table 1: The maximum h_0, h_1, and h_2 accuracy achieved for each data set across varying top-k orderings and brute-force attack.

	User	Top-1			Top-3			Top-15			Top-50			Top-100			Top-500			Random		
		h_0	h_1	h_2	h_0	h_1	h_2	h_0	h_1	h_2	h_0	h_1	h_2	h_0	h_1	h_2	h_0	h_1	h_2	h_0	h_1	h_2
E-mail	**1**	0.15	0.38	0.53	0.19	0.42	0.54	0.61	0.65	0.66	0.61	0.65	0.66	0.62	0.65	0.66	0.63	0.65	0.66	0.06	0.16	0.30
	2	0.15	0.40	0.66	0.17	0.40	0.66	0.65	0.73	0.77	0.66	0.74	0.78	0.67	0.74	0.78	0.68	0.74	0.78	0.06	0.18	0.34
	3	0.12	0.40	0.51	0.17	0.40	0.56	0.21	0.49	0.58	0.40	0.55	0.59	0.40	0.55	0.59	0.40	0.55	0.59	0.10	0.17	0.33
R5	**1**	0.04	0.31	0.56	0.09	0.43	0.66	0.29	0.71	0.82	0.49	0.80	0.84	0.65	0.84	0.84	0.76	0.84	0.84	0.06	0.20	0.46
	2	0.08	0.19	0.35	0.17	0.37	0.53	0.23	0.46	0.54	0.31	0.50	0.54	0.38	0.53	0.54	0.50	0.53	0.54	0.08	0.21	0.46
	3	0.03	0.26	0.44	0.13	0.38	0.50	0.25	0.50	0.63	0.30	0.52	0.64	0.41	0.55	0.64	0.46	0.60	0.65	0.11	0.24	0.46
R10	**1**	0.10	0.39	0.60	0.23	0.67	0.81	0.46	0.85	0.89	0.59	0.89	0.89	0.71	0.89	0.89	0.87	0.89	0.89	0.09	0.23	0.45
	2	0.06	0.27	0.46	0.14	0.47	0.63	0.29	0.61	0.71	0.37	0.68	0.73	0.44	0.69	0.73	0.53	0.72	0.73	0.06	0.23	0.45
	3	0.07	0.27	0.44	0.11	0.40	0.54	0.24	0.52	0.62	0.31	0.57	0.64	0.37	0.60	0.64	0.43	0.61	0.65	0.07	0.23	0.46
R15	**1**	0.06	0.31	0.51	0.13	0.47	0.64	0.25	0.58	0.71	0.40	0.69	0.73	0.46	0.70	0.73	0.55	0.70	0.74	0.05	0.16	0.34
	2	0.09	0.34	0.54	0.17	0.44	0.63	0.26	0.60	0.67	0.39	0.66	0.67	0.44	0.66	0.67	0.54	0.67	0.67	0.07	0.19	0.35
	3	0.07	0.38	0.57	0.20	0.56	0.73	0.33	0.67	0.78	0.45	0.75	0.80	0.54	0.76	0.80	0.62	0.76	0.81	0.07	0.18	0.35

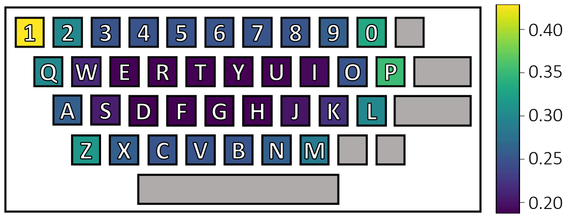


Figure 12: With the dynamically positioned keyboard defense, the probability for each key that the adversary's guess on the key is a true positive. Outward keys have more chance to be correctly guessed since they have fewer target positions after displacement that conflict with the other keys.

it preserves the keyboard layout and only alters the position of the keyboard one to two keys away from the original position.

5 Related Work

Keylogging attacks have been studied for a long time, although in different contexts. The rise in the popularity of AR/VR systems drew the attention of the community to study the security and privacy of AR/VR, including the potential risks of novel text input methods.

Attacks on AR/VR Keyboards. The built-in system keyboards in current AR/VR HMDs share a similar layout to the AR keyboard we studied in this paper. However, targeting and selection methods vary. Kreider [26] explores the feasibility of keylogging inference attacks on the HoloLens keyboard, which targets by the headset and selects by tapping gesture. The adversary obtains the drawmetric profiles of 10 possible passwords in advance and infers which one the victim types by analyzing the profile extracted from the victim's video recording. Although the study is based on manual processing, which is impractical for random length inputs from unknown, intractably large input sets, the evidence underlines the potential privacy leakage due to the visual side-channel for AR/VR HMDs. On the other hand, our method is a systematic approach that works on random inputs of any length. Arafat et al. [10] performed a keylogging inference attack on VR HMDs by observing the patterns in CSI distortions. They were able to obtain up to 69% of the virtual keystrokes. Ling et al. [30] demonstrate computer vision- and motion sensor-based attacks to infer passwords for two modes: (1) targeting via headset and selecting via controller and (2) targeting and selecting via a controller. Both approaches rely on the assumption that the keyboard is fixed in position and the size in the virtual environment in order to use pre-computed rotation angles for inference. On the contrary, we account for the possibility that the virtual objects can be freely positioned and scaled in the virtual environment by the user, which otherwise limits the flexibility of virtual applications.

Other Keylogging Inference Attacks. Since the first keylogging inference attack over 50 years ago [13, 18], there have been many incidents that emerged or potential risks underlined by researchers in the field [37]. Among those works, ours falls in the cluster of studies where spatial information is utilized to detect the key locations and achieve inference without user profiling. Sun et al. [42] present an approach for inference from tablet backside motion patterns available with video recordings. They utilize top-k and h-hop to evaluate their results, pointing that high 1- and 2-hop accuracy values indicate a significant information gain. Jin et al. [21] used the desk motion patterns to achieve inference from video recordings. They represented words with a sequence of group labels by replacing each character with its group label, where a group stands for a set of neighbor keys in some proximity. Using a Long Short Term Memory technique over sequences of such groups to obtain words, they improved the accuracy, showing the importance of accurate 1- or 2-hop granularity.

6 Limitations and Future Work

Our work has several limitations, which we list in the following.

Contextualized Learning. For long text inputs, the results show that the adversary achieved 65% pinpoint accuracy by inspecting as few as 15 reconstructions. Since such inputs are likely to include context, the accuracy could be further improved through deduction. In our future work, we will utilize language models to systematically achieve this task, which could also help highlight reconstructions with context to reduce the search space for the adversary.

Short Inputs. The attack accuracy reaches up to 87% for short text inputs, which provides a significant information gain to the adversary. In short messages with context, the language models could be very beneficial. On the other hand, in login credentials, the language context might be abstracted from the text. It is expected that such short texts would not follow the regular language models, leaving all reconstructions equally probable to the adversary.

Other Modalities. Comparing the results of the key identification stage with that of the end-to-end pipeline, we observe that the main stage hindering reaching a 100% pinpoint accuracy is the keystroke detection. We aim to improve the keystroke detection by exploiting other methods using additional clues in our future work. More specifically, we intend to explore to what extend the eye and head movements could contribute to the keystroke detection phase.

7 Conclusion

In this paper, we presented a keylogging inference attack targeting in air tapping keyboards for AR/VR HMDs by exploiting the observation that hands follow specific patterns while users are typing in the air. Substantiated by three different attack scenarios and threat models with reasonable capabilities, our attack provides up to 87% accuracy in inferring a random text of any length without requiring any special user profiling. We discuss various countermeasures to the attack, and show that they are a nontrivial task as they often conflict with usability. In our future work, we will exploit language models to increase accuracy, optimize the attack, and further evaluate the trade-off between usability and defenses.

Acknowledgement. This work was supported in part by NRF grant 2016K1A1A2912757 (GRL) and CyberFlorida Seed Award (2020).

References

[1] Nefes data kit untethers USB devices for wireless VR setups. `https://www.tomshardware.com/news/nefes-data-kit-wireless-vr,34162.html`, Apr. 2017. Accessed: 2020-03-12.

[2] Keyboard mixed reality. `https://docs.microsoft.com/en-us/windows/mixed-reality/design/keyboard`, 2019. Accessed: 2021-05-07.

[3] Augmented reality company builds rapid deployment kits in response to covid-19, Mar 2020.

[4] Magic leap 1 — magic leap. `https://www.magicleap.com/magic-leap-1`, 2020. Accessed: 2020-03-13.

[5] Microsoft hololens — mixed reality technology for business. `https://www.microsoft.com/en-us/hololens`, 2020. Accessed: 2020-03-13.

[6] Raptor - everysight. `https://everysight.com/product/raptor/`, 2021. Accessed: 2021-12-31.

[7] Thinkreality a3. `https://www.lenovo.com/us/en/thinkrealitya3`, 2021. Accessed: 2021-12-31.

[8] Tracking — leap motion controller. `https://www.ultraleap.com/product/leap-motion-controller/`, 2021. Accessed: 2021-05-07.

[9] Vuzix blade. `https://www.vuzix.com/products/vuzix-blade-smart-glasses-upgraded`, 2021. Accessed: 2021-12-31.

[10] A. A. Arafat, Z. Guo, and A. Awad. Vr-spy: A side-channel attack on virtual key-logging in vr headsets. In *2021 IEEE Virtual Reality and 3D User Interfaces (VR)*, pp. 564–572, 2021. doi: 10.1109/VR50410.2021.00081

[11] R. Azuma, Y. Baillot, R. Behringer, S. Feiner, S. Julier, and B. MacIntyre. Recent advances in augmented reality. *IEEE Comput. Graph. Appl.*, 21(6):34–47, Nov. 2001. doi: 10.1109/38.963459

[12] R. T. Azuma. A survey of augmented reality. *Presence: Teleoper. Virtual Environ.*, 6(4):355–385, Aug. 1997. doi: 10.1162/pres.1997.6.4.355

[13] D. Boak. A history of us communications security. 1973.

[14] C. Cao, Z. Li, P. Zhou, and M. Li. Amateur: Augmented reality based vehicle navigation system. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.*, 2(4), Dec. 2018. doi: 10.1145/3287033

[15] J. Carmigniani, B. Furht, M. Anisetti, P. Ceravolo, E. Damiani, and M. Ivkovic. Augmented reality technologies, systems and applications. *Multimedia Tools and Applications*, 51:341–377, 2010.

[16] S. Dash. Here's how indian startup blinkin helped set up acs in hospitals in wuhan during the coronavirus crisis, Mar 2020.

[17] A. Doshi, S. Y. Cheng, and M. M. Trivedi. A novel active heads-up display for driver assistance. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 39(1):85–93, 2009. doi: 10.1109/TSMCB.2008.923527

[18] J. Friedman. Tempest: A signal problem. 35:76, 1972.

[19] Y. Gan, T. Wang, A. Javaheri, E. Momeni-Ortner, M. Dehghani, M. Hosseinzadeh, and R. Rawassizadeh. 11 years with wearables: Quantitative analysis of social media, academia, news agencies, and lead user community from 2009-2020 on wearable technologies. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.*, 5(1), Mar. 2021. doi: 10.1145/3448096

[20] Y. Gu, C. Yu, Z. Li, Z. Li, X. Wei, and Y. Shi. Qwertyring: Text entry on physical surfaces using a ring. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.*, 4(4), Dec. 2020. doi: 10.1145/3432204

[21] K. Jin, S. Fang, C. Peng, Z. Teng, X. Mao, L. Zhang, and X. Li. Vivisnoop: Someone is snooping your typing without seeing it! In *2017 IEEE Conference on Communications and Network Security (CNS)*, pp. 1–9, Oct 2017. doi: 10.1109/CNS.2017.8228624

[22] D. Kariuki. Hypergrid business, Feb 2020.

[23] K. Kim, M. Billinghurst, G. Bruder, H. B. Duh, and G. F. Welch. Revisiting trends in augmented reality research: A review of the 2nd decade of ismar (2008–2017). *IEEE Transactions on Visualization and Computer Graphics*, 24(11):2947–2962, Nov 2018. doi: 10.1109/TVCG.2018.2868591

[24] D. P. Kingma and J. Ba. Adam: A method for stochastic optimization. *CoRR*, abs/1412.6980, 2014.

[25] B. Klimt and Y. Yang. The enron corpus: A new dataset for email classification research. In *Proceedings of the 15th European Conference on Machine Learning*, ECML'04, p. 217–226. Springer-Verlag, Berlin, Heidelberg, 2004. doi: 10.1007/978-3-540-30115-8-22

[26] C. Kreider. The discoverability of password entry using virtual keyboards in an augmented reality wearable: an initial proof of concept. In *SAIS*, 2018.

[27] K. Lebeck, K. Ruth, T. Kohno, and F. Roesner. Towards security and privacy for multi-user augmented reality: Foundations with end users. In *2018 IEEE Symposium on Security and Privacy (SP)*, pp. 392–408, May 2018. doi: 10.1109/SP.2018.00051

[28] H. Li, A. Gupta, J. Zhang, and N. Flor. Who will use augmented reality? an integrated approach based on text analytics and field survey. *European Journal of Operational Research*, 10 2018. doi: 10.1016/j.ejor.2018.10.019

[29] J.-W. Lin, P.-H. Han, J.-Y. Lee, Y.-S. Chen, T.-W. Chang, K.-W. Chen, and Y.-P. Hung. Visualizing the keyboard in virtual reality for enhancing immersive experience. In *ACM SIGGRAPH 2017 Posters*, SIGGRAPH '17. Association for Computing Machinery, New York, NY, USA, 2017. doi: 10.1145/3102163.3102175

[30] Z. Ling, Z. Li, C. Chen, J. Luo, W. Yu, and X. Fu. I know what you enter on gear vr. In *2019 IEEE Conference on Communications and Network Security (CNS)*, pp. 241–249, June 2019. doi: 10.1109/CNS.2019.8802674

[31] F. Lu and D. A. Bowman. Evaluating the potential of glanceable ar interfaces for authentic everyday uses. In *2021 IEEE Virtual Reality and 3D User Interfaces (VR)*, pp. 768–777, 2021. doi: 10.1109/VR50410.2021.00104

[32] I. S. Mackenzie and S. X. Zhang. An empirical investigation of the novice experience with soft keyboards. *Behaviour & Information Technology*, 20:411–418, 2001.

[33] A. Maiti, M. Jadliwala, and C. Weber. Preventing shoulder surfing using randomized augmented reality keyboards. In *2017 IEEE International Conference on Pervasive Computing and Communications Workshops (PerCom Workshops)*, pp. 630–635, March 2017. doi: 10.1109/PERCOMW.2017.7917636

[34] D. Maloney. Augmented reality aids in the fight against covid-19, Mar 2020.

[35] R. McPherson, S. Jana, and V. Shmatikov. No escape from reality: Security and privacy of augmented reality browsers. In *Proceedings of the 24th International Conference on World Wide Web*, WWW '15, p. 743–753. International World Wide Web Conferences Steering Committee, Republic and Canton of Geneva, CHE, 2015. doi: 10.1145/2736277.2741657

[36] Z. Medenica, A. L. Kun, T. Paek, and O. Palinko. Augmented reality vs. street views: A driving simulator study comparing two emerging navigation aids. In *Proceedings of the 13th International Conference on Human Computer Interaction with Mobile Devices and Services*, MobileHCI '11, p. 265–274. Association for Computing Machinery, New York, NY, USA, 2011. doi: 10.1145/2037373.2037414

[37] J. V. Monaco. Sok: Keylogging side channels. In *2018 IEEE Symposium on Security and Privacy (SP)*, pp. 211–228, May 2018. doi: 10.1109/SP.2018.00026

[38] L. Pavanatto, C. North, D. A. Bowman, C. Badea, and R. Stoakley. Do we still need physical monitors? an evaluation of the usability of ar virtual monitors for productivity work. In *2021 IEEE Virtual Reality and 3D User Interfaces (VR)*, pp. 759–767, 2021. doi: 10.1109/VR50410.2021.00103

[39] P. A. Rauschnabel, R. Felix, and C. Hinsch. Augmented reality marketing: How mobile AR-apps can improve brands through inspiration. *Journal of Retailing and Consumer Services*, 49(C):43–53, 2019. doi: 10.1016/j.jretconser.2019

[40] K. Ruth, T. Kohno, and F. Roesner. Secure multi-user content sharing for augmented reality applications. In *Proceedings of the 28th USENIX Conference on Security Symposium*, p. 141–158, 2019.

[41] P. Shirley and S. Marschner. *Fundamentals of Computer Graphics*. A. K. Peters, Ltd., USA, 3rd ed., 2009.

[42] J. Sun, X. Jin, Y. Chen, J. Zhang, Y. Zhang, and R. Zhang. Visible: Video-assisted keystroke inference from tablet backside motion. In *NDSS*, 2016. doi: 10.14722/ndss.2016.23060

[43] K. Sun, W. Wang, A. X. Liu, and H. Dai. Depth aware finger tapping on virtual displays. In *Proceedings of the 16th Annual International Conference on Mobile Systems, Applications, and Services*, MobiSys '18, p. 283–295. Association for Computing Machinery, New York, NY, USA, 2018. doi: 10.1145/3210240.3210315

[44] D. Takahashi. Pokémon go is the fastest mobile game to hit $600 million in revenues, Oct 2016.

[45] M. Tonnis, C. Lange, and G. Klinker. Visual longitudinal and lateral driving assistance in the head-up display of cars. In *IEEE and ACM International Symposium on Mixed and Augmented Reality*, pp. 91–94, 2007. doi: 10.1109/ISMAR.2007.4538831

[46] M. Wilson. Snap is the world's most innovative company of 2020, Mar 2020.

[47] K. Wolf, K. Marky, and M. Funk. We should start thinking about privacy implications of sonic input in everyday augmented reality! 2018.

[48] Y. Yin, Q. Li, L. Xie, S. Yi, E. Novak, and S. Lu. Camk: Camera-based keystroke detection and localization for small mobile devices. *IEEE Transactions on Mobile Computing*, 17(10):2236–2251, 2018. doi: 10.1109/TMC.2018.2798635

[49] C. Yu, Y. Gu, Z. Yang, X. Yi, H. Luo, and Y. Shi. Tap, dwell or gesture? exploring head-based text entry techniques for hmds. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, CHI '17, p. 4479–4488. Association for Computing Machinery, New York, NY, USA, 2017. doi: 10.1145/3025453.3025964

[50] C. Yu, K. Sun, M. Zhong, X. Li, P. Zhao, and Y. Shi. One-dimensional handwriting: Inputting letters and words on smart glasses. In *Proceedings of the 2016 CHI Conference on Human Factors in Computing Systems*, CHI '16, p. 71–82. Association for Computing Machinery, New York, NY, USA, 2016. doi: 10.1145/2858036.2858542

[51] F. Zhang, Z. Liu, Z. Cheng, O. Deussen, B. Chen, and Y. Wang. Mid-air finger sketching for tree modeling. In *2021 IEEE Virtual Reality and 3D User Interfaces (VR)*, pp. 826–834, 2021. doi: 10.1109/VR50410.2021.00110

[52] H. Zhang, Y. Yin, L. Xie, and S. Lu. Airtyping: A mid-air typing scheme based on leap motion. In *Adjunct Proceedings of the 2020 ACM International Joint Conference on Pervasive and Ubiquitous Computing and Proceedings of the 2020 ACM International Symposium on Wearable Computers*, UbiComp-ISWC '20, p. 168–171. Association for Computing Machinery, New York, NY, USA, 2020. doi: 10.1145/3410530.3414387