Virtual Keymysteries Unveiled: Detecting Keystrokes in VR with External Side-Channels

Hossein Khalili[1], Alexander Chen[1], Theodoros Papaiakovou[1], Timothy Jacques[1], Hao-Jen Chien[1]
Changwei Liu[2], Aolin Ding[2], Amin Hass[2], Saman Zonouz[3], Nader Sehatbakhsh[1]

[1]SsysArch Lab, ECE Department, University of California, Los Angeles, CA, USA
[2]Cyber Lab, Accenture Labs, Accenture Cyber Fusion Center, Arlington, Virginia, USA
[3]CPSec Lab, SCP Department, Georgia Institute of Technology, Atlanta, GA, USA

***Abstract*—AR/VR devices are becoming prevalent, permeating different facets of our daily lives. Nevertheless, this prevalence presents fresh security and privacy hurdles as users increasingly employ these devices to manage sensitive data such as passwords, personal information, and financial data in potentially *insecure* settings. Due to these concerns, there has been an increasing trend in the literature to analyze security and privacy threats for AR/VR by proposing novel attack strategies. While effective and worrisome, the existing body of work has focused mostly on *internal* threats for AR/VR devices, such as malicious sensors, apps, or firmware. However, in this paper, we focus on a new facet of this body of research by designing an *external* attacker. The key observation is that although the virtual world remains concealed from an external observer (i.e., an adversary), the physical interactions required to input commands into the VR world are observable and create a *side channel*. Building upon this finding, we conduct a practical attack, named `LensHack`, on Quest 2 VR devices. By employing our algorithm and an external camera (Blink), we capture and analyze the interactions between the user and the device, successfully extracting typed characters with over 80% accuracy.**

I. Introduction

AR/VR devices have become commonplace and integrated into many parts of our lives. They provide engaging and interactive experiences, changing the way we approach several fields such as gaming, education, healthcare, and entertainment. With wearable headsets and mobile apps, AR/VR devices are increasingly accessible, improving our interactions with digital content and transforming our perspectives on the world [1], [2], [3], [4], [5], [6].

The surge in AR/VR device popularity, however, brings forth unavoidable security and privacy vulnerabilities as users handle sensitive data via these devices. Consequently, akin to prevailing computing models, AR/VR devices are prone to diverse software, network, mobile, and sensor attacks. Indeed, a comprehensive array of attacks has already been directed at numerous aspects of AR/VR devices [7], [8], [9], [10], [11], [12]. This emerging attack category is highly domain-focused, capitalizing on distinctive AR/VR device traits (e.g., seamless sensor-computation fusion, diverse sensor modalities, etc.).

Existing attack scenarios, however, have mainly focused on addressing *internal* threats to VR/AR devices. This refers to situations where an adversary has somehow infiltrated the system, such as its firmware, software, sensors, or other combinations. As a result, they can initiate diverse forms of confidentiality-integrity-availability and/or privacy attacks [13], [14], [9]. In contrast to this existing line of research, as shown recently [15], [16], AR/VR devices are also distinctively susceptible to an *external* attacker, where a bystander (a human attacker and/or a digital camera) can record and dissect a user's (victim's) interactions with the AR/VR device. This allows for the extraction of sensitive data like keystrokes, passwords, and more. The comparison aligns with external attacks on different computing models (e.g., IoT devices) where an attacker sniffs packets or executes a physical side-channel attack.

For assessing the practicality of this attack in real-life situations, we create `LensHack`, a new attack approach for deriving keystrokes via analyzing recorded frames from an external camera. In Section II, we initially outline our threat model and assumptions, followed by an in-depth explanation of the attack strategy in Section III.

We evaluate our attack strategy in real-world setups using a Meta Quest 2 VR device and a small indoor security camera (Amazon Blink). We study the robustness of `LensHack` under different setups (distance and angle).

In short, the contributions of this paper are as follows:

- We present a new attack strategy, `LensHack`, that leverages an external camera to record videos and analyzes the videos to infer keystrokes.
- A proof-of-concept implementation of the attack using Meta Quest 2 under various configurations.

II. Problem Statement

Overview. For a more systematic assessment of potential dangers, a typical indoor setup for an AR/VR device is illustrated in Figure 1. AR/VR devices commonly have a headset and hand controllers (B, C). They may be wired to an interface box or even a PC (D, E). Alternatively, they might interact with a tracking pad (F), a camera (A), or a PC (E). The room might have other external sensors such as a security camera (A) and/or routers or computers (E). Given the provided setup, three categories of attacks exist:

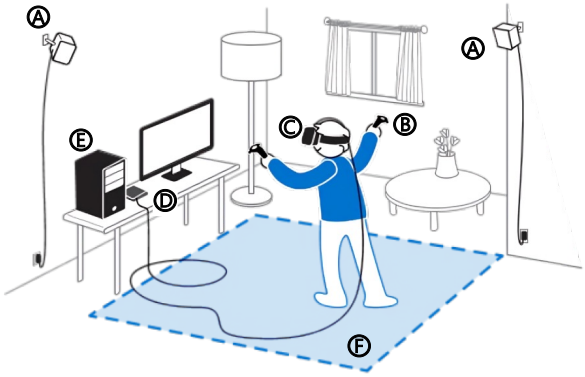


Fig. 1: Main components of an AR/VR system (A) cameras, (B) controllers, (C) VR headset, (D) interface box (if needed), (E) PC and/or router, (F) tracked workspace (if needed).

- ***(Apps/Software)*** The most prevalent type of attack on AR/VR devices is when a malicious app is co-located with the victim's device [9], [17], [18], [19], [20]. The goal of the app is typically either stealing some sensitive information and/or causing integrity and/or availability attacks. The malicious app usually infiltrates the device using common techniques for installing malware. Once it is installed, the app is activated and executes its malicious payload (usually in the background and stealthy).
- ***(Hardware/Sensor)*** Another common type of attack on AR/VR devices is to manipulate its hardware and/or OS/firmware [21], [11], [12], [13], [22]. Considering that AR/VR devices heavily depend on diverse sensors, adversaries are afforded a broad spectrum of chances to target these sensors. Similar to existing sensor attacks, the attack is domain-specific to maximize success.
- ***(External)*** The attack vector proposed in this work is an external one where a security camera and/or other sensors unrelated to the AR/VR device but present in the same room can extract sensitive information from the device and particularly the interaction between the user and the device. The important feature of this attack is that the attacker does not need to infiltrate and/or install any app on the AR/VR device and the attack is completely external.

Threat Model. We assume an adversary who tries to infer the keystrokes inserted by a victim interacting with an AR/VR device. We assume that the device is completely secure, that there is no malicious application installed, that the firmware/OS is unmodified, and that the hardware components and sensors work correctly without bugs. Instead, we assume a completely *external* attack model. Specifically, the adversary is capable of observing the user (victim) while they are interacting with the AR/VR device. This is achieved, for example, by collecting video frames from the interaction between the user and the device using a camera. However, we assume the adversary has no control over the virtual world and cannot observe any inputs and/or commands that are purely in the virtual domain.

We envision two possibilities for our attack: *(a)* the victim is in a public place (e.g., metro, coffee shop, etc.) and is interacting with their AR/VR device, hence the adversary can get close and record the frames using their own device. *(b)* The victim is in-door but the adversary is either using a hidden camera that is pre-installed (e.g., an Airbnb, a hotel room, etc.), or hacking into an existing security camera, or the adversary leverages a long-range high-quality lens to record the frames from a long distance (e.g., a window across the street). In Section IV, we discuss our setup and how each assumption impacts our design, attack, and results. We also discuss the impact of the camera's angle, distance, and other obstacles that might protect the user's hands.

III. LensHack: Attack Strategy

In short, `LensHack` leverages an external camera, an Amazon Blink in our implementation, to record the frames when a victim is interacting (typing) with an AR/VR device (Meta Quest 2), and a multi-step algorithm to analyze the frames to infer the keystrokes.

The core of our design is a six-step algorithm which is depicted in Figure 2. Briefly, `LensHack` first extract the 3D key points (basic features) from each frame ❶. These points are then fed into a module for detecting "clicks" to recognize when a new key is pressed ❷. Simultaneously, using the 3D key points, various features are extracted to estimate the approximate location of the keyboard ❸-❹. Using the location of the keyboard and hands, our algorithm then estimates the keystrokes and outputs a word ❺. Further, using a new technique, `LensHack` adjusts itself based on the new observations to improve the overall accuracy ❻. In the following, we describe each step in more detail.

1) 3D feature extraction using MediaPipe. Video frames are first fed into a feature extraction block to identify the hand, body location, and direction. We opt for using MediaPipe as the feature extractor. Google's MediaPipe is a popular machine-learning model for image processing and has been used for a variety of vision tasks including hand tracking, object detection, motion tracking, etc. In our setup, MediaPipe is particularly configured to output two sets of data: *(a)* hand index points and *(b)* body index points.

Precisely, MediaPipe yields 21 2D points (x,y) for every hand and 33 for the complete body. This outcome is termed *raw features* and is represented as F_{rh}, F_{lh}, and F_b for the right hand, left hand, and whole body, respectively. For a deeper understanding of MediaPipe's internal structure and its functioning, we direct readers to the research conducted by Lugaresi *et al.* [23].

2) Click (Keystroke) Detection. The second step for the attack is detecting when a key is pressed. Given that our framework continuously analyzes the frames, it is important to find the key press "event" accurately and timely.

To find the key press, which we call "click" in this paper, we observe that the distance between the index finger and thumb can accurately reveal the event. More specifically, if the distance between the index and thumb is monitored (using

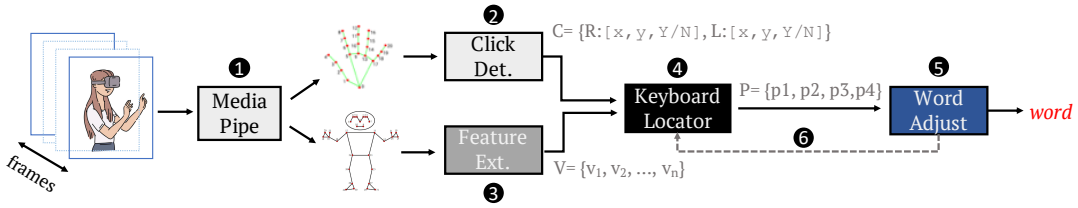


Fig. 2: The main steps for LensHack. Our framework takes a frame, extracts the hand and body locations, detects the keystrokes (clicks), localizes the keyboard based on the body and hand locations, and extracts the characters for each keystroke.

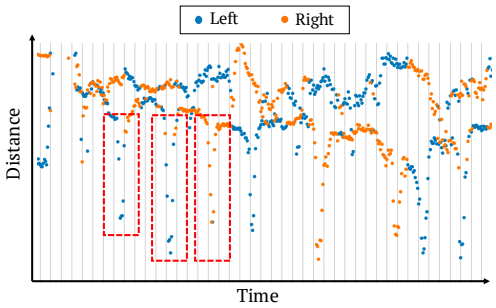


Fig. 3: The distance between the index finger and thumb is shown for both hands. Local minima indicate a click (key press) as highlighted by the rectangles.

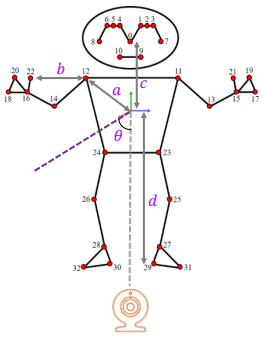


Fig. 4: To estimate the location of the keyboard, we first extract multiple different features (shown as 'a'-'d' and θ) from the raw features created by MediaPipe.

the relevant indices in F_{rh} and F_{lh}), *valleys* will indicate a click. Such observation is further depicted in Figure 3 where the distance between two fingers for each hand is shown over time and clicks are highlighted using rectangles.

Care should be taken to detect the valleys (local minima). As can be seen in Figure 3, the distance signal is fairly noisy as the user constantly moves their hands/fingers and hence the distance moves up and down over time. As a result, simple detection of minima would result in high false positives. Instead, we observe that the peak's *prominence* is a very robust indicator of the click as only true clicks result in very prominent spikes in the signal shown in Figure 3. Using this insight, LensHack first computes the distance between relative raw features collected from step ❶, and then first detects local minima and then computes the prominence for each spike. Using trial and error, we then set a threshold, P_{th}, and label a spike as click only if its prominence is larger than this threshold.

It is important to mention that a click often lasts multiple frames (depending on the frame rate and type speed). To avoid false keystroke detection, it is important to properly filter click events as only one click per keystroke should be detected. As a result, only one frame out of consecutive T frames should be analyzed at the later stages of the framework. This is achieved by using a buffer and outputting only one element of the buffer while discarding the rest (the discarding is done when the event is finished – i.e., when no click was detected for five consecutive frames). We observe that our algorithm is not sensitive to which frame to use, thus we choose the middle frame as the representative (although using any other frame had very similar results based on our initial analysis).

3) (Advanced) Feature Extraction. In parallel with click detection, our framework analyzes the raw features to extract more (advanced) features. As we will describe later, these features will be used to precisely estimate the location of the keyboard in the space.

For keyboard location estimation, we observe that the location is dependent on multiple factors hence the features should be carefully selected to maximize the accuracy.

In particular, we note that the positioning of the keyboard is inherently linked to the camera's perspective and the user's bodily alignment. Additionally, we ascertain that the keyboard's placement is influenced by the user's physical attributes, primarily their height, the gap between their arm and fingers, the separation between their arm and head, and the midpoint between their body and the ground.

Using these insights, we extract five main features shown in Figure 4. It is important to mention that we considered other potential features including the distance between arms and heads, etc. but did not find a statistically significant correlation between these other features and the keyboard location.

4) Estimating the Keyboard Location. Using the selected frame that includes a click (step ❷) and the extracted features from the 3D index points in step ❸, the next major step is to estimate the location of the keyboard to ultimately infer the exact keystroke (i.e., which key has been pressed).

We have already discussed in the previous step that the key-

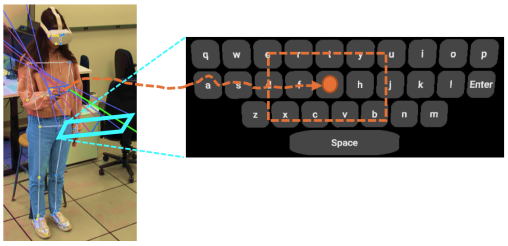


Fig. 5: Steps required to locate the keyboard and inferring the key. Our framework leverages the features described in Figure 4 to first locate the keyboard (the estimated location is shown in light blue). Using the location, known layout, and the fingers' location, our algorithm locates the exact key (e.g., letter 'g' in this example).

board's location is a function of the user's physical attributes. Using the features explained in Figure 4, the "Keyboard Locator" block, outputs the location of the keyboard in space. To achieve this, we explored various solutions including a rule-based method and machine learning-based estimation. Using various experiments, we ultimately develop a machine learning-based location estimator.

Specifically, our block takes a frame, the location of fingers and the (advanced) features derived from step ❸ and outputs a vector of four 2D (x,y) points ($V = \{v_1, v_2, v_3, v_4\}$). The points indicate the four corners of the keyboard deciding its exact location and size. Note that, we assume that the adversary already knows the victim's keyboard layout hence once the location and size are known, the exact location of each key on the keyboard is calculated by the block.

Internally, our machine learning block to estimate the location is a fully connected neural network (multi-layer perceptron or MLP) consisting of three layers (an input and output layer and a hidden layer). The input layer has the same size as the number of features, while the output layer has eight outputs formed into four pairs (for each 2D point representing a corner of the keyboard). We varied different sizes for the hidden layer, ultimately finding that a layer with ten neurons achieves the best result.

Once the location of each individual key is known, the final and most critical step is to infer the key. This is done by checking the coordinates of the finger and mapping it into the predicted location. In Figure 5, we illustrate these steps in more detail.

5) Adjusting the Keyboard Location. While the location of the key is determined by this point, we propose an additional step to increase the accuracy of the attack further.

Specifically, we observe that our keyboard location detection algorithm on average has up to two 'hop' errors on the x-axis and one 'hop' error on the y-axis. For a QWERTY keyboard, we define *hop* as the characters that are within one key distance of the pressed key. For example, for an x-axis hop, a 2-hop error means that for character 'g', the potential outcomes are {'d', 'f', 'g', 'h', 'j'}. Similarly, for the 1-hop y-axis, the options are one row above or below (if any) the pressed key.

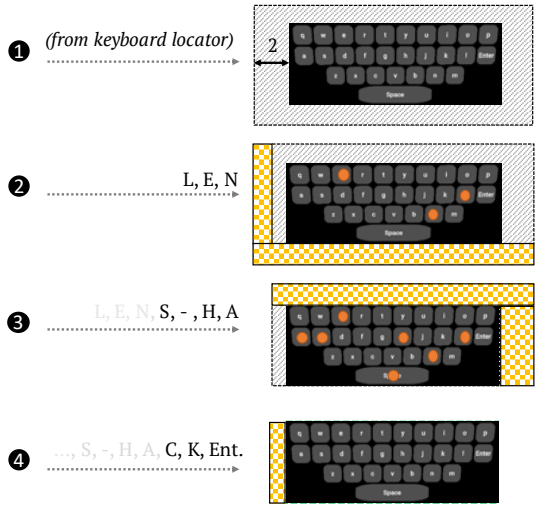


Fig. 6: How the *Word Adjust* block can find the exact spot of the keyboard after a few keystrokes. The figure shows how the possible locations of the keyboard change as the word "Lens Hack" is being typed.

Considering this intrinsic error, two choices emerge. Initially, there's the possibility of emphasizing the enhancement of location estimation. Alternatively, there's the option of prioritizing *adjusting* the location as more keys are observed by our framework.

During the design of our location estimator (❹), we emphasized heavily improving the accuracy. However, we observe that the online adjustment is needed to achieve high accuracy. As a result, we design a "Word Adjust" block where the goal is to reduce the detection error by reducing the search space as more inputs (key press) are observed.

The fundamental concept of how this block operates is depicted in Figure 6. Overall, recall that our location estimator has a 2-hop x-axis and 1-hop y-axis error. Nevertheless, upon pressing a new key, the *unauthorized* positions become eliminable, thus reducing the search space. For instance, the figure illustrates the transformation in the search space while typing the word "Lens Hack". The initial trio of characters occupies distinct rows, resulting in the y-axis search space narrowing to two alternatives (either as displayed or shifted up by one row). The downward shift ceases to be viable as it would misplace the 'E' character into an unauthorized location — that is, beyond the projected rectangle. Furthermore, these three characters also eliminate one possibility along the x-axis. Continuing the typing, more possibilities are eliminated as the user types characters such as 'Space', 'A', and 'Enter'. Evenetually, as the last character is pressed, only one viable location remains allowing to achieve 100% accuracy in keystroke detection for this particular example.

There are two important considerations. First, in order to achieve better accuracy, our algorithm dynamically adjusts the location of the keyboard (step ❻). After each adjustment, the algorithm also dynamically adjusts its previous predictions. Collectively, this means that the accuracy can significantly

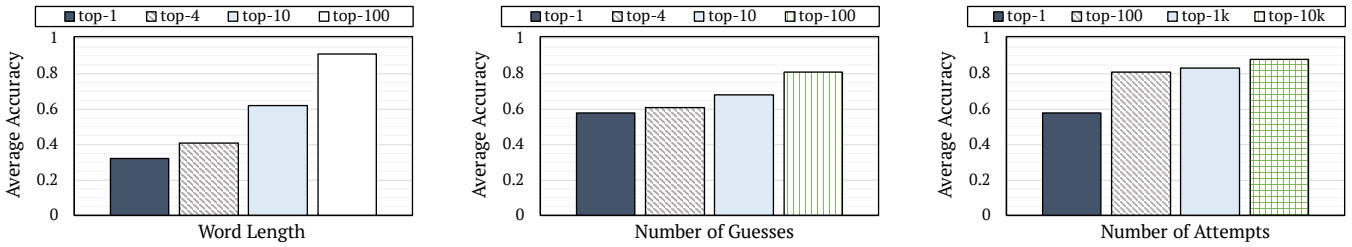


Fig. 7: Results for `LensHack`. (a) Accuracy per character. The value of top-n specifies the number of characters (n) entered by the user. (b) Accuracy per word. Each word is between 6-12 characters. The result shows the accuracy of top-n guesses made by our framework. (c) Accuracy per word when combined with a sophisticated brute-force strategy. Our algorithm randomly shifts the keyboard location and adds/removes characters. Top-n represents the number of attempts by the algorithm to guess the password. Results demonstrate orders of magnitude fewer attempts are needed to achieve high accuracy.

increase for ALL characters and not just for *future* characters.

The second consideration is that 100% accuracy is not always achievable. The reason for this is that the victim's word(s) may not be long enough to eliminate all the possibilities. We will show how the accuracy changes when the word size changes in Section IV.

IV. Evaluations

Setup. To show the feasibility of the proposed attack, we run various experiments using different configurations. To collect data, we employ a Meta Quest 2 alongside an Amazon Blink security camera with 30 fps (frame per second). The camera rests on a tripod positioned on a desk, resulting in an approximate total setup height of three meters. To record the samples, we use an Amazon Blink Sync Module and flash drive. Using the Blink mobile app, we then manually record the videos. As described in Section II, during a real attack, the assumption is that the attacker either directly controls the camera and/or hacks into the cloud account and hence is able to capture/download the videos.

To input the words, the user stands in front of the camera, about four meters away from it. To study the impact of angle, we use three configurations: *angle*=$\{0, 45, 75\}$. We did not investigate the effect of long distances, however, our preliminary experiment showed that the accuracy does not change in a room setting (i.e., when the distance is $<$10m).

For data collection, we have five individuals (all co-authors of the paper) positioned before the camera, tasked with inputting a group of words. These words are chosen from a Kaggle dataset containing prevalent passwords and their associated strengths [24]. For each user, 100 different passwords are randomly selected. The user then types these words one by one. Each video is separately recorded and labeled with appropriate tags (username, configuration, and word). We did not intentionally speed up and/or slow down the typing process and each user was instructed to type the words with their normal typing speed. Our further analysis, however, showed that the typing speed did not have any impact on the accuracy.

Results. We report the results for the attack scenario in Figure 7. All results are the average accuracy across five users, typing 100 different words for different configurations. We repeat each experiment for three configurations (i.e., three different camera angles).

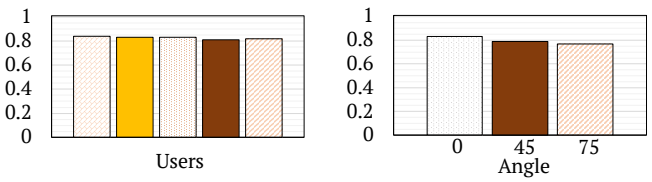
We report the results in three different categories. In Figure 7a, the accuracy of character detection is shown. The results are presented in four different categories: *top-1*, *top-4*, *top-10*, and *top-100*. Recall that as we explained in Step ❺ and ❻ in Section III, our algorithm adaptively improves its detection accuracy by observing more frames and intelligently eliminating the impossible configurations. As a result, we expect to see improved results as more characters are in. To capture this, the four different categories are selected where *top-x* indicates that 'x' keys have been observed so far (note that the keys are not necessarily different).

As can be seen from Figure 7a, the accuracy significantly improves as more characters are seen by our framework. Specifically, the per-character accuracy with no adjustments is about 35%. However, this number quickly grows as the number of characters increases to four, ten, and then ultimately a hundred. At its maximum, we observe more than 80% per-character accuracy.

To further examine the accuracy of `LensHack`, we run a new experiment where the user is typing a password that is 6-12 characters long. We then use our framework to predict the password and report the accuracy in Figure 7b. Note that adjusting is only applied while typing each word and not before or after (i.e., for each word, we reset the configuration to make each inference independent).

The results in Figure 7b are shown in four categories where top-x represents the first x predictions made (each answer is created by taking the predicted location of the keyboard and then moving it randomly in either the x- or y-axis). The findings reveal an accuracy increase from 55% to 80% when permitting the algorithm to make additional guesses.

Finally, we delve deeper into analyzing the effect of employing a more intricate brute-force algorithm on accuracy. We note that, aside from inaccuracies stemming from projecting the keyboard's position, another notable contributor to inaccuracy lies in click identification—specifically, instances of



(a) (b)

Fig. 8: Sensitivity analysis for LensHack under various settings. (a) Accuracy per word with brute-force for five different users. (b) Accuracy per word with brute-force when the camera is at an angle in relation to the user.

overestimating and underestimating clicks (where the assumption might be that the password has a length of n characters when it is actually $n \pm r$). Consequently, our advanced brute-force approach not only systematically explores various x- and y-axis shifts of the keyboard, but also introduces random additions or removals of characters from the password at various locations of the password. The outcomes are depicted in Figure 7c. Analogous to the preceding figure, distinct setups indicate varying attempt counts. The findings illustrate that employing a more advanced brute-force attack can yield accuracy levels approaching 90%.

To explore the influence of diverse measurement setup factors on accuracy, we depict the outcomes in Figure 8, showcasing the variations in camera angle and per-user accuracy reporting. The results for both diagrams correspond to the brute-force test (refer to Figure 7c). Accuracy experiences only marginal alteration in both scenarios. The fluctuation among users in Figure 8a is approximately $\pm$ 3%, signifying the algorithm's limited sensitivity to user behavior and/or physical traits. We observe that the location accuracy is slightly impacted by the person's physical attributes, however, that impact is pretty low.

For different angles shown in Figure 8b, the differences are slightly higher. As the camera is more angled, the accuracy drops. However, we observe that this drop is less than 5% at most. We also contrast the outcomes with the camera positioned on the left and right sides of the user, yet discern no notable distinction.

V. Related Work

Software Attacks. The most common type of attack on AR/VR devices is a software attack [9], [25], [17], [18], [19], [26], [20]. In this category, the adversary manages to install a malicious app on the AR/VR device and then leverages this application to perform a malicious action.

Specifically, several works focused on extracting keys by running a background malicious application [9]. For example, Zhang *et al.* [9] proposed a new attack model to extract keystrokes by using various available side-channel information including the frame rate and memory. A common defense relies on enforcing standard techniques for isolation and information flow [27], [28], [29], [30], [31], [32].

Hardware/Sensor Attack. The other possible class of attacks on AR/VR devices are sensor and/or hardware-based attacks [21], [11], [12], [13], [22], [5]. In this class, the goal is to attack the sensors to create various malicious behaviors and/or exploit them to extract sensitive information either from the device itself or even from the surrounding environment. Also related are methods that exploit the VR device network traffic to extract information [33], [34].

Close to this work is the method proposed by Ling *et al.* [11], where the user's head motions were monitored to extract the keystrokes. Further, the authors presented a model to extract the keystrokes when the user is using a gear to insert the keys. Also close to this work is an attack method proposed by Meteriz *et al.* [10] where the key taps were detected externally by exploiting a motion sensor connected to the AR/VR device. In contrast to these studies, our attack model addresses a more challenging context by exclusively utilizing raw video frames for hand movement extraction, without resorting to lasers, motion sensors, or specialized gear-generated signals. Moreover, our approach is more accurate with only a limited observation of characters, in contrast to prior approaches that necessitate a larger sample size for comparable precision.

Similarly, Luo *et al.* [12], Slocum *et al.* [7], and Wu *et al.* [8] proposed new methods to exploit motion sensors to infer keys. Compared to these methods, LensHack relies on a stronger threat model where no malicious application needs to be installed on the device and the attack is completely external and independent of the device.

While preparing the final version of this work, the authors became aware of a very recent work that achieved similar capabilities to LensHack. This complementary exploration proposed by Gopal *et al.* [15] delves into comparable avenues to ours and addresses similar concerns regarding the security and privacy implications of AR/VR devices in the context of user interactions and an external adversary. By adopting an analogous perspective, the manuscript investigates a novel attack vector and its potential impact on user data confidentiality. While both works address a similar problem, there are several important differences between our approaches. First, our approach is independently developed and utilizes a completely different strategy for finding the location of the keyboard. As explained in Section III, we extract advanced features from the 3D key points and leverage a machine-learning model to estimate the keyboard's location. Gopal *et al.*, conversely, relies on prior information about the camera location to estimate the keyboard and further relies on observing many characters to find the exact location. As we showed in this paper, by only observing 5-10 characters, our method can achieve up to 80% accuracy.

VI. Conclusions

In this paper, we presented a new attack strategy for inferring keystrokes on AR/VR devices. The key insight was to leverage an external observer to extract the information about the user's interaction. We showed that by analyzing the

recorded videos collected by a camera, one can successfully extract the keys pressed by the user in the virtual domain.

Given the increasing prominence of AR/VR devices, there is a pressing need to conduct comprehensive and thorough examinations of the potential risks and hazards associated with the usage of such devices.

REFERENCES

[1] T. Zhang, C. Shi, P. Walker, Z. Ye, Y. Wang, N. Saxena, and Y. Chen, "Passive vital sign monitoring via facial vibrations leveraging ar/vr headsets," in Proceedings of the 21st Annual International Conference on Mobile Systems, Applications and Services, 2023, pp. 96–109.

[2] L. He, H. Hou, S. Shi, X. Shuai, and Z. Yan, "Towards bone-conducted vibration speech enhancement on head-mounted wearables," in Proceedings of the 21st Annual International Conference on Mobile Systems, Applications and Services, 2023, pp. 14–27.

[3] V. Nair, G. M. Garrido, and D. Song, "Exploring the unprecedented privacy risks of the metaverse," arXiv preprint arXiv:2207.13176, 2022.

[4] S. Wang, S. Yang, H. Li, X. Zhang, C. Zhou, C. Xu, F. Qian, N. Wang, and Z. Xu, "Salientvr: saliency-driven mobile 360-degree video streaming with gaze information," in Proceedings of the 28th Annual International Conference on Mobile Computing And Networking, 2022, pp. 542–555.

[5] C. Shi, X. Xu, T. Zhang, P. Walker, Y. Wu, J. Liu, N. Saxena, Y. Chen, and J. Yu, "Face-mic: inferring live speech and speaker identity via subtle facial dynamics captured by ar/vr motion sensors," in Proceedings of the 27th Annual International Conference on Mobile Computing and Networking, 2021, pp. 478–490.

[6] S. Eom, M. Hadziahmetovic, M. Pajic, and M. Gorlatova, "Through an ar lens: Augmented reality magnification through feature detection and matching," in Proceedings of the 20th ACM Conference on Embedded Networked Sensor Systems, 2022, pp. 784–785.

[7] C. Slocum, Y. Zhang, N. Abu-Ghazaleh, and J. Chen, "Going through the motions:{AR/VR} keylogging from user head motions," in 32nd USENIX Security Symposium (USENIX Security 23), 2023, pp. 159–174.

[8] Y. Wu, C. Shi, T. Zhang, P. Walker, J. Liu, N. Saxena, and Y. Chen, "Privacy leakage via unrestricted motion-position sensors in the age of virtual reality: A study of snooping typed input on virtual keyboards," in 2023 IEEE Symposium on Security and Privacy (SP). IEEE Computer Society, 2023, pp. 3382–3398.

[9] Y. Zhang, C. Slocum, J. Chen, and N. Abu-Ghazaleh, "It's all in your head (set): Side-channel attacks on ar/vr systems," in USENIX Security, 2023.

[10] Ü. Meteriz-Yıldıran, N. F. Yıldıran, A. Awad, and D. Mohaisen, "A keylogging inference attack on air-tapping keyboards in virtual environments," in 2022 IEEE Conference on Virtual Reality and 3D User Interfaces (VR). IEEE, 2022, pp. 765–774.

[11] 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). IEEE, 2019, pp. 241–249.

[12] S. Luo, X. Hu, and Z. Yan, "Hologlogger: Keystroke inference on mixed reality head mounted displays," in 2022 IEEE Conference on Virtual Reality and 3D User Interfaces (VR). IEEE, 2022, pp. 445–454.

[13] M. Corbett, B. David-John, J. Shang, Y. C. Hu, and B. Ji, "Bystandar: Protecting bystander visual data in augmented reality systems," in Proceedings of the 21st Annual International Conference on Mobile Systems, Applications and Services, 2023, pp. 370–382.

[14] Y. Chen, T. Li, R. Zhang, Y. Zhang, and T. Hedgpeth, "Eyetell: Video-assisted touchscreen keystroke inference from eye movements," in 2018 IEEE Symposium on Security and Privacy (SP). IEEE, 2018, pp. 144–160.

[15] S. R. K. Gopal, D. Shukla, J. D. Wheelock, and N. Saxena, "Hidden reality: Caution, your hand gesture inputs in the immersive virtual world are visible to all!" in 32nd USENIX Security Symposium (USENIX Security 23), 2023, pp. 859–876.

[16] S. Luo, A. Nguyen, H. Farooq, K. Sun, and Z. Yan, "Eavesdropping on controller acoustic emanation for keystroke inference attack in virtual reality," in The Network and Distributed System Security Symposium (NDSS), 2024.

[17] K. Ruth, T. Kohno, and F. Roesner, "Secure {Multi-User} content sharing for augmented reality applications," in 28th USENIX Security Symposium (USENIX Security 19), 2019, pp. 141–158.

[18] J. Shang, S. Chen, J. Wu, and S. Yin, "Arspy: Breaking location-based multi-player augmented reality application for user location tracking," IEEE Transactions on Mobile Computing, vol. 21, no. 2, pp. 433–447, 2020.

[19] L. S. Figueiredo, B. Livshits, D. Molnar, and M. Veanes, "Prepose: Privacy, security, and reliability for gesture-based programming," in 2016 IEEE Symposium on Security and Privacy (SP). IEEE, 2016, pp. 122–137.

[20] H. Farrukh, R. Mohamed, A. Nare, A. Bianchi, and Z. B. Celik, "{LocIn}: Inferring semantic location from spatial maps in mixed reality," in 32nd USENIX Security Symposium (USENIX Security 23), 2023, pp. 877–894.

[21] J. R. Sanchez Vicarte, B. Schreiber, R. Paccagnella, and C. W. Fletcher, "Game of threads: Enabling asynchronous poisoning attacks," in Proceedings of the Twenty-Fifth International Conference on Architectural Support for Programming Languages and Operating Systems, 2020, pp. 35–52.

[22] J. O'Hagan, P. Saeghe, J. Gugenheimer, D. Medeiros, K. Marky, M. Khamis, and M. McGill, "Privacy-enhancing technology and everyday augmented reality: Understanding bystanders' varying needs for awareness and consent," Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies, vol. 6, no. 4, pp. 1–35, 2023.

[23] C. Lugaresi, J. Tang, H. Nash, C. McClanahan, E. Uboweja, M. Hays, F. Zhang, C.-L. Chang, M. G. Yong, J. Lee et al., "Mediapipe: A framework for building perception pipelines," arXiv preprint arXiv:1906.08172, 2019.

[24] B. BANSAL. (2023) Password strength classifier dataset. [Online]. Available: https://www.kaggle.com/datasets/bhavikbb/password-strength-classifier-dataset?resource=download

[25] T. Kohno, J. Kollin, D. Molnar, and F. Roesner, "Display leakage and transparent wearable displays: Investigation of risk, root causes, and defenses," Microsoft Research, Tech. Rep., February 2015.[Online]. Available https ..., Tech. Rep.

[26] J. Hu, A. Iosifescu, and R. LiKamWa, "Lenscap: split-process framework for fine-grained visual privacy control for augmented reality apps," in Proceedings of the 19th annual international conference on mobile systems, applications, and services, 2021, pp. 14–27.

[27] Y. Kim, S. Goutam, A. Rahmati, and A. Kaufman, "Erebus: Access control for augmented reality systems," in 32nd USENIX Security Symposium (USENIX Security 23), 2023, pp. 929–946.

[28] F. Roesner, D. Molnar, A. Moshchuk, T. Kohno, and H. J. Wang, "World-driven access control for continuous sensing," in Proceedings of the 2014 ACM SIGSAC conference on computer and communications security, 2014, pp. 1169–1181.

[29] 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). IEEE, 2018, pp. 392–408.

[30] R. Herbster, S. DellaTorre, P. Druschel, and B. Bhattacharjee, "Privacy capsules: Preventing information leaks by mobile apps," in Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services, 2016, pp. 399–411.

[31] K. Olejnik, I. Dacosta, J. S. Machado, K. Huguenin, M. E. Khan, and J.-P. Hubaux, "Smarper: Context-aware and automatic runtime-permissions for mobile devices," in 2017 IEEE Symposium on Security and Privacy (SP). IEEE, 2017, pp. 1058–1076.

[32] N. Raval, A. Razeen, A. Machanavajjhala, L. P. Cox, and A. Warfield, "Permissions plugins as android apps," in Proceedings of the 17th Annual International Conference on Mobile Systems, Applications, and Services, 2019, pp. 180–192.

[33] R. Trimananda, H. Le, H. Cui, J. T. Ho, A. Shuba, and A. Markopoulou, "{OVRseen}: Auditing network traffic and privacy policies in oculus {VR}," in 31st USENIX security symposium (USENIX security 22), 2022, pp. 3789–3806.

[34] A. Al 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). IEEE, 2021, pp. 564–572.