MotionDecipher: General Video-assisted Passcode Inference In Virtual Reality

Guanchong Huang
University of Oklahoma
Norman, OK, USA
guanchong.huang@ou.edu

Yan He
University of Oklahoma
Norman, OK, USA
heyan@ou.edu

Shangqing Zhao
University of Oklahoma
Norman, OK, USA
shangqing@ou.edu

Yi Wu
University of Oklahoma
Norman, OK, USA
yi.wu-1@ou.edu

Song Fang*
University of Oklahoma
Norman, OK, USA
songf@ou.edu

***Abstract*—Virtual Reality (VR) technology offers a portable and immersive experience at users' fingertips. Personal Identification Numbers (PINs) are essential for accessing VR devices and applications, such as shopping, banking, and payments. While recent studies infer VR keystrokes using sensors such as inertial measurement units (IMUs) in headsets, they require pre-infecting devices with malware, limiting real-world applicability. Non-invasive approaches have emerged, yet they primarily target traditional keyboards, which are visible to all and operated directly by hand. In contrast, VR keyboards are virtual, visible only to the user, and operated via controllers, rendering traditional methods ineffective. This paper presents *MotionDecipher*, a novel video-assisted attack that infers a victim's PIN input in VR devices from footage of hand and controller movements during typing. *MotionDecipher* captures both controller-triggered keypress events in the time domain and inter-key movement trajectories in the spatial domain. Varying PINs may result in distinguishable spatiotemporal patterns, which an attacker can capture and analyze to recover the input. Experimental results on three commercial VR devices validate attack efficacy across different environmental conditions.**

***Index Terms*—Virtual Reality, PIN Authentication, Keystroke Inference, Spatiotemporal Analysis**

I. Introduction

The Virtual Reality market was valued at USD 13.58 billion in 2023 and is projected to grow from USD 16.05 billion in 2024 to USD 62.39 billion by 2032, exhibiting a compound annual growth rate (CAGR) of 18.5% from 2024 to 2032 [1]. The number of VR users in the United States was 73.3 million in 2023 and is forecasted to reach 91.3 million by 2028 [2]. People use VR for education, gaming, entertainment, shopping, and to join the metaverse that is a virtual world where individuals can interact even when they are not physically in the same place [3]. To access these VR devices or various provided services (such as payments, content unlocking, or parental controls), users inevitably need to enter sensitive information, such as Personal Identification Numbers (PINs), for authentication or authorization. For instance, many VR devices, including Meta Quest 3 [4] and HTC VIVE Focus 3 [5], implement PINs as a quick, familiar method for securing access to them. Consequently, these sensitive PINs become natural targets for attackers.

Emerging research efforts have proposed techniques to infer keystrokes in VR environments (e.g., [6], [7], [8], [9]), while they are all invasive and require deceiving the target VR device to pre-install malware. Such malware is designed to intentionally access and transmit sensitive data from various sensors embedded within VR devices, such as IMU sensors (accelerometers and gyroscopes) or hand tracking sensors. These sensors capture critical information correlated with the victim's keystrokes. For example, a user's head moves in subtle ways as she types, and IMU sensors may track head movements [9]; hand tracking sensors can reveal the positions and angles of the user's hands and fingers during password entry [7]. The captured data are then sent to a remote site, where an adversary can analyze it to infer the keystrokes.

Recording the typing process with cameras provides a non-invasive solution to infer keystrokes on traditional keypads. This technique leverages visual cues from a user's hand movement [10], [11], [12], eye movement [13], screen reflection [14], tablet backside motion [15], or shadow formation around the fingertip [16] to reconstruct the input keystrokes without requiring physical access to the device. Such vision-based techniques all assume that the user presses keys on a clearly visible, stationary keyboard (either physical or touchscreen) with fingers. This setup allows cameras to capture the key-specific hand, finger, eye, or device motion information, which can then be analyzed to infer keystrokes. However, in VR scenarios, both the keyboard display and the user's eyes are enclosed within the headset "shell", creating a unique, isolated environment where only the user can see the virtual keyboard or interface they're interacting with. Thus, keystroke input in VR brings two new challenges that undermine the effectiveness of traditional vision-based keystroke inference techniques. First, the user's eyes and input keyboard are invisible to anyone outside, making video-based attacks (e.g., [13], [14]) that leverage eye movements or screen reflections impractical. Second, rather than using direct pressing action, users often operate one or two controllers,

*Corresponding author

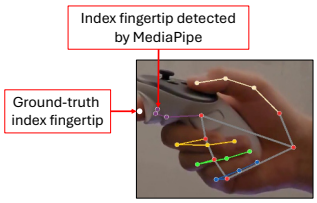


Fig. 1: MediaPipe fails to track the index fingertip.

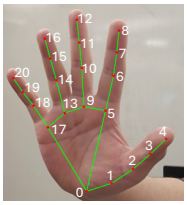


Fig. 2: Hand landmarks' indices (0-20).

each equipped with buttons, to cast virtual laser rays to select keys on virtual keyboards that float in space, disrupting the spatial consistency typically used to correlate finger or hand positions with specific keys (e.g., [10], [11], [12]).

Due to the unique setup and constraints of VR environments, no known vision-based techniques can recover keystrokes inputted via VR controllers. Only two recent non-invasive, non-vision-based research efforts [17], [18] aim to infer controller-based keystrokes in VR. Particularly, the wireless signal based technique [17] exploits the variations in the captured WiFi channel state information (CSI) to recognize the virtual keystrokes. However, this method requires deliberate placement of both a transmitter and a receiver, positioned such that the victim falls within the direct line between the two devices. Also, CSI is highly sensitive to environmental changes and may fluctuate significantly in response to small variations, such as people walking nearby or interference from other moving body parts. Moreover, user-specific training is required to account for variations in VR typing patterns, such as duration and motion. The acoustic emanation based work [18] infers the keystrokes by placing a recording smartphone surrounding the victim and eavesdropping on the clicking sounds of the moving hand controller during keystrokes. However, due to the fast attenuation of acoustic signals over distance [19], this method only works in a limited sensing range (e.g., within 2 meters), and the inference accuracy greatly decreases as the recording distance increases. The limited sensing range of acoustic sensing [20] presents significant hurdles for applying acoustic emanation-based schemes in practice.

In this paper, we present the design and evaluation of *MotionDecipher*, a novel and practical attack that infers a victim's PIN input on VR devices by deciphering video footage of their hand-controller interaction during PIN entry. *MotionDecipher* is motivated by the observation that keystrokes at different positions on the virtual PIN pad cause the user's hand (or controller) to exhibit distinct self-contained motion patterns, which can be extracted and leveraged to infer the typed PIN.

The design of *MotionDecipher* faces two major challenges. First, the typing finger is wrapped around the controller, and the motion caused by pulling the trigger is subtle and embedded within broader hand movements. This makes it challenging to recognize when the user inputs a digit. Intuitively, the state-of-the-art hand tracking tools, e.g., MediaPipe [21] that track on-hand keypoints, may offer a solution. Unfortunately, in controller-based input, the occlusion caused by the controller significantly reduces the accuracy of hand movement tracking. For instance, as shown in Fig. 1, although the index fingertip keypoint remains visible, MediaPipe incorrectly detects it with a substantial displacement from the ground truth, making it unreliable for determining whether the trigger has been pressed. To address this, we design a two-layer algorithm to recognize keypresses. In the first layer, we train an instance segmentation model to isolate the region containing only the hand and controller. In the second layer, we build a classification model to identify keypress events based on the segmented input. The second challenge lies in mapping the captured keystroke frames to the actual digits of the typed PIN. As mentioned earlier, the virtual PIN pad is not visible from the attacker's perspective, making it impossible to directly associate the finger's position during input with specific key locations. Accordingly, we develop a spatiotemporal correlation algorithm to translate a sequence of keystroke-associated frames into the corresponding PIN.

To prevent attackers from continuously trying different PINs, a VR system can enforce an exponential backoff strategy, where the system waits for increasingly longer periods after each failed PIN attempt. Also, the system can implement an account lockout mechanism that temporarily disables PIN entry after a certain number of consecutive failures (e.g., 5 or 10 attempts). Both strategies can significantly limit an attacker's ability to cycle through a list of inferred PINs. However, even when such defenses are in place, *MotionDecipher* remains highly impactful: by narrowing the candidate PIN set, it substantially increases the probability of success within the limited number of allowed attempts.

The PIN retry and backoff behavior on Meta Quest devices is undocumented in any official public source. Empirically, we observe that Meta Quest 2 and 3 implement a tiered PIN retry mechanism combining fixed and escalating delays. The system permits five initial failed attempts without delay, followed by a 30-second pause before the sixth. Attempts 6 through 10 proceed without further delay. A second 30-second delay occurs before the 11th attempt, after which each failure through the 39th incurs a fixed 30-second delay. Beyond that, delays increase progressively: 60 seconds for attempts 40 through 49, 120 seconds for 50 through 59, 240 seconds for 60 through 69, and 480 seconds for 70 through 79. This coarse-grained backoff strategy delays escalation and permits dozens of PIN guesses without imposing prohibitive wait time. Such leniency likely reflects Meta's emphasis on usability over strict security enforcement and further underscores the timeliness and practical importance of *MotionDecipher*.

In summary, our main contributions are as follows:

- We propose *MotionDecipher*, the first video-based technique to infer PINs entered via VR controllers.
- We design a holistic deep learning-based framework to recognize keypress events from recordings of hand-controller interactions during PIN entry.
- We develop an algorithm that deciphers the typed PIN by analyzing the spatiotemporal motion patterns embedded in the sequence of keystroke-associated frames.

- We carry out extensive real-world experiments, demonstrating that *MotionDecipher* can consistently and significantly reduce the search space for each PIN.

II. Preliminaries

A. MediaPipe

MediaPipe [21], developed by Google, is a state-of-the-art hand-tracking tool that tracks 21 keypoints corresponding to the joints of each hand, as shown in Fig. 2. Notably, MediaPipe provides precompiled Python packages [22] for ease of use, but Google does not release the source code or training data for the underlying deep neural networks (DNNs) used in models. This limitation prevents users from fine-tuning, retraining, or modifying the core model architecture and weights with their own data. Consequently, we cannot directly utilize MediaPipe for VR keystroke behavior detection.

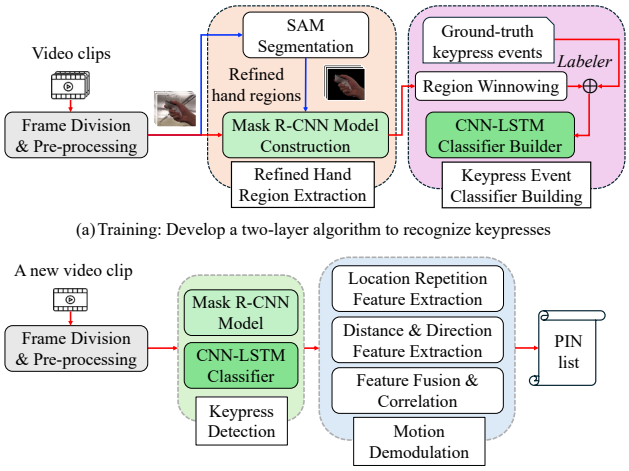
B. Mask R-CNN and CNN-LSTM

Object detection involves identifying and localizing objects in an image or video by drawing bounding boxes around them. Regions with Convolutional Neural Network (CNN) features (R-CNN) [23] is an early and foundational object detection method. Faster R-CNN [24] improves speed and scalability over the original R-CNN. Mask R-CNN [25] builds directly on Faster R-CNN, with additional components to predict object masks, enabling instance segmentation, where the pixel-wise boundaries of each object instance can be identified. In this study, we build a Mask R-CNN model to eliminate the background around the hand area and achieve high keypress event recognition accuracy.

The Convolutional Neural Network-Long Short-Term Memory Network (CNN-LSTM) [26], [27] is a hybrid deep learning architecture that combines the distinct advantages of CNNs and Long Short-Term Memory Networks (LSTMs), a specialized type of Recurrent Neural Networks (RNNs) [28]. CNNs are well-suited for spatial feature extraction [29], [30], while LSTMs excel at handling sequential data [31]. Unlike traditional RNNs, which suffer from the gradient vanishing problem (i.e., the gradients used to update the weights during training become extremely small during backpropagation) when processing long sequences and can only capture dependencies in short sequences, LSTM incorporates memory cells and gating mechanisms (i.e., input, forget, and output gates) to selectively control the information flow, capturing complex dependencies over extended time frames. CNN-LSTM can extract complex spatiotemporal features, making it particularly suitable for tasks involving tracking object changes across a long continuous video frame stream.

III. Adversary Model

We consider a general scenario, where a user uses a controller to input a PIN to unlock a VR device or for authentication in applications such as purchases [32]. Typically, when a user puts on a VR headset, they must type the PIN before turning it on or waking it up. The study employs a standard 10-digit passcode input on the virtual keypad of a VR device. An adversary stealthily records the typing scenario with a single RGB camera (e.g., a smartphone camera) from a distance of at least 3 meters from the user, and aims to infer the typed PIN by processing the recorded video. The recording captures the hand and controller movement during typing, and does not need to capture the victim's whole body.



(a) Training: Develop a two-layer algorithm to recognize keypresses

(b) Inference: Infer the PIN with a two-stage procedure

Fig. 3: Workflow of *MotionDecipher*.

IV. Attack Design

A. Attack Overview

Fig. 3 illustrates the high-level design of *MotionDecipher*, consisting of two phases, i.e., the *training* and *inference*.

Fig. 3a depicts the offline training phase, in which a two-layer algorithm is developed to recognize keypresses. The attacker first divides the recorded video into frames, and pre-processes them to coarsely extract hand movement data (i.e., the recorded raw hand regions containing hands, controllers, and the environment). The pre-processed frames input into a module called Segment Anything Model (SAM) [33] to separate refined hand regions (containing only hands and controllers) from the background area for training a Mask R-CNN model, which helps automatically and accurately extract refined hand regions per pre-processed video frame. Such regions, together with their corresponding keypress events (keypress or non-keypress) are then utilized to train a CNN-LSTM classifier. In the inference phase, as shown in Fig. 3b, after dividing the recorded video into frames and pre-processing them, the attacker utilizes two important modules to achieve PIN inference: *keypress detection* and *motion demodulation*. For the initial module, the trained Mask R-CNN model and CNN-LSTM classifier are utilized successively to detect keypress events. In the second module, the attacker extracts the temporal feature from a detected sequence of individual keypresses, then extracts the spatial distance and direction features. Finally, all extracted features are combined into a spatiotemporal representation that correlates with a digit sequence, enabling PIN inference.

B. Training Phase

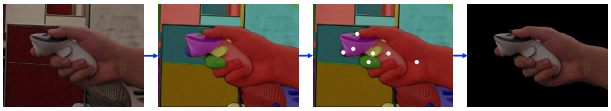
1) Frame Division and Pre-processing: We first divide the recorded video clip into individual raw video frames using OpenCV, an open-source library for computer vision and image processing [34]. Since the region showing the hand typically occupies only a small portion of the entire raw video frame, we use object detection tools (e.g., MediaPipe) to automatically crop the target hand region, resulting in a rectangular bounding box around the hand holding the controller for each frame where the hand is detected.

2) Refined Hand Region Extraction: The background in the pre-processed frames may adversely affect keypress event detection accuracy. We train a Mask R-CNN model to extract the refined hand region (containing only the hand and the controller it operates). SAM (Segment Anything Model), developed by Meta AI, is applied to perform fine-grained segmentation to get refined hand regions, and correct segmentation results will be used to train the Mask R-CNN model.

SAM Segmentation: Fig. 4 illustrates how SAM segmentation works. With a pre-processed frame in Fig. 4a, SAM first identifies the edges of various objects in it, and generates masks with different colors for each detected object, as illustrated in Fig. 4b. We then manually select the desired regions containing the hand and the controller by clicking on the areas of interest. As shown in Fig. 4c, those manual clicks are marked with small white dots. Fig. 4d shows the resultant refined hand region. In the output frame, all pixels not belonging to the refined hand region are set to black.

Mask R-CNN Model Construction: SAM-assisted refined hand region extraction has two disadvantages that hinder its scalability and reliability. First, it lacks the capability to automatically extract masks for specific objects, relying instead on manual selection, significantly reducing efficiency. Second, SAM may introduce segmentation errors and fail to recognize the hand and controller correctly. Therefore, we propose to train a Mask R-CNN model to automatically extract refined hand regions in pre-processed frames with high accuracy. When SAM segmentation outputs correct results, such data (i.e., pre-processed frames and corresponding refined hand regions) are utilized to train the Mask R-CNN model.

We record videos of a user operating the controller of a Meta Quest 3 to input various digits (0–9). The user can freely adjust their hand positions and orientations while entering digits. Each video segment capturing the typing of a single digit lasts 2 seconds to cover the entire keypress process. With a frame rate of 60 frames per second (FPS), this results in $10 \times 2 \times 60 = 1,200$ raw video frames in total. With SAM segmentation, we obtain 1,094 correctly segmented refined hand regions. These 1,094 corresponding pre-processed frames and their associated refined hand regions constitute the training dataset. We apply the standard 80/20 split for creating the training and test sets, ResNet50 [35], [25] as the backbone, and stochastic gradient descent (SGD) [36] for optimization. Also, Appendix A shows example cases where Mask R-CNN is able to extract refined hand regions while SAM fails.



(a) Preprocessed frame (b) SAM-generated masks (c) Mask selection (d) Refined hand region

Fig. 4: Refined hand region segmentation leveraging SAM.

3) Keypress Event Classifier Building: Capturing keypresses in VR (i.e., trigger presses) requires integrating spatial and temporal information. Spatial details identify the typing finger's position and movement, while temporal information tracks the sequence and timing of the keypress: approach, press, and release. Thus, accurately recognizing and analyzing keypresses necessitates a model capable of understanding both the spatial and temporal features of the keypress process. Accordingly, we choose CNN-LSTM, a supervised learning (classification) technique that can effectively capture both spatial and temporal features in video streams, as detailed in Section II-B, for recognizing keypress and non-keypress events with using the refined hand regions. Manual annotation is carried out to build the training dataset for training a CNN-LSTM keypress event classifier. This process involves labeling each input frame containing only refined hand region with the corresponding keypress event (keypress or non-keypress) to provide the ground truth data needed for supervised learning.

Region Winnowing: Occasionally, the constructed Mask R-CNN model may have extraction errors. We thus perform a consistency check to see whether the extracted refined hand regions are correct. We winnow out incorrectly recognized refined hand regions, and the rest regions become training data for building a CNN-LSTM classifier. We form a dataset containing refined hand regions from 10 individuals (3 females and 7 males) wearing Meta Quest 3 headsets. We take 50 videos for every person to record their hand-controller interaction. The video resolution is 1080p (1920×1080 pixels) with a frame rate of 60 FPS, which is typical for smartphones. Each video lasts 2 seconds, where a user uses a controller to input a random digit. Thus, we have a total of $50 \times 10 \times 2 \times 60$=60,000 frames. After region winnowing, we obtain 59,056 correctly identified refined hand regions, inputted to event labeling.

Event Labeling: With refined hand regions, *MotionDecipher* creates a training dataset by labeling keypress events at the frame level. For the obtained 59,056 frames, 52,285 are labeled as non-keypress, while 6,771 are labeled as keypress. The common 80:20 train-test split is used to divide the training dataset for building the CNN-LSTM model.

CNN-LSTM Classifier Builder: We aim to build a model to detect the number of individual keypresses, i.e., the PIN length, by recognizing keypress events from frames. Each keypress generates a frame stream that captures the gradual movement of the finger as it presses and releases the key. An individual keypress is considered successfully recognized if at least one keystroke-associated frame during its duration is detected. Meanwhile, if two detected keystroke-associated frames are separated by fewer than 10 frames, they are considered part of the same individual keypress, as an individual

TABLE I: Comparison of CNN with Mask R-CNN, CNN-LSTM without Mask R-CNN, and CNN-LSTM with Mask R-CNN in terms of the number of detected keypress events. We abbreviate "Mask R-CNN" as "M".

Users	CNN w/ M	CNN-LSTM w/o M	CNN-LSTM w/ M
P1	0/50	1/50	46/50
P2	3/50	0/50	47/50
P3	2/50	0/50	48/50
P4	0/50	2/50	46/50
P5	4/50	0/50	45/50

keypress typically generates at least 10 frames empirically.

To enhance the generalization of the trained model, data augmentation is applied to expand the variety of training data. Particularly, during training, we utilize two preprocessing layers (i.e., tf.keras.layers.RandomFlip [37] and tf.keras.layers.RandomRotation [38]) in TensorFlow Keras API to randomly flip and rotate frames, respectively. In addition, we implement learning rate decay [39] and early stopping [40] during the training process to prevent overfitting, and employ Adam [41] for optimization. Table I reveals the keypress events detection accuracy for 5 previously unknown users, each of whom generates 50 keypress events. The CNN-LSTM with Mask R-CNN achieved an average keypress detection accuracy of 92.8%, significantly outperforming CNN with Mask R-CNN (3.6%) and CNN-LSTM without Mask R-CNN (1.2%). These results demonstrate the necessity of using CNN-LSTM with Mask R-CNN for keypress event detection.

C. Inference Phase

The adversary first performs frame division and preprocesses the obtained frames as in the training phase. With the pre-processed frames, the built Mask R-CNN model and CNN-LSTM classifier will be applied sequentially to achieve keypress detection. Next, a motion demodulation module is employed to extract spatiotemporal features associated with PIN inputs and translate them into PINs.

1) Motion Demodulation: We aim to identify a shared feature that links a sequence of individual keypresses in a recorded video to a passcode, enabling its inference. A user may move their hand (controller) while typing different digits. We define *keypress location* as the position of the typing finger within the corresponding video frame when a keypress is detected, which varies depending on the digit typed. However, as the attacker cannot observe the virtual keypad, such keypress locations cannot be directly mapped into typed digits. We aim to explore a feature to characterize the spatiotemporal structure of a detected keypress sequence in the recorded video. Ideally, this feature can uniquely determine a passcode. For a $l-$digit passcode, there exist $T_{max} = 10^l$ possibilities in total. Thus, in an ideal scenario, a perfect feature should divide T_{max} candidates into T_{max} subsets. Each subset would then hold only one passcode corresponding to the observed feature.

We utilize the identified feature to divide all T_{max} candidates into T subsets. To quantify the distinguishability of this feature, we define a metric - *feature distinction rate*, denoted as $\eta = \frac{T}{T_{max}}$ ($\eta \in (0, 1]$ as $T \leq T_{max}$). When η approaches 1, we generate more subsets, implying a higher degree of differentiation achieved using the feature. Therefore, our goal is to identify a feature with a maximum η, representing the highest ability to distinguish among candidates.

Location Repetition Feature Extraction: Intuitively, except for distinguishing PIN length, we can determine whether or not two individual keypresses over different times are the same. If two keypress locations are identical or highly similar, it indicates that they correspond to the same digit. For an N-digit passcode, we denote its N corresponding keypress locations as $l_1, l_2, \cdots, l_N$. We represent its location repetition as $\mathcal{T} = [r_1, r_2, \cdots, r_N]$, referred to as *temporal vector*. To get the value of the vector $\mathcal{T}$, we initialize $r_1 = 1$ and set $i = 2$, then proceed through the following steps:

(1) if l_i and l_j ($j < i$) are identical or highly similar, then $r_i = r_j$; otherwise, $r_i = max(r_1, r_2, \cdots, r_{i-1}) + 1$, where max is a function that returns the maximum value among all parameters.

(2) we increment i and jump to step (1), and stop until $i = N$.

For example, the passcode '260260' would yield its temporal vector of $[1, 2, 3, 1, 2, 3]$. This is because the first three digits are distinct, and the last three digits repeat the first three. Using the location repetition of a 6-digit passcode, we can divide all 10^6 possible combinations into 203 distinct groups, where all members within a group share the same $\mathcal{T}$ value. On average, each group contains approximately $10^6/203 \approx 4,926$ passcodes. Consequently, an input keypress sequence is mapped to one of these 4,926 passcodes based on this feature. Therefore, the feature distinction rate is $203/10^6 = 2.03 \times 10^{-4}$.

The location repetition feature (i.e., temporal feature) only indicates whether any two digits in different positions are the same, without accounting for spatial features, i.e., the specific changes between successive digits if they differ. Intuitively, when the user presses two different keys, the movement distance of the laser point is proportional to the inter-key distance (*IKD*), which refers to the center-to-center spacing between keys on the virtual keypad. Meanwhile, the movement direction of the laser point varies according to the inter-key direction (*IKR*), which represents the direction of the following key relative to the current key. In return, the *IKD* and *IKR* can facilitate narrowing down the candidates for the pair of successive keypresses. In the following, we evaluate the distinction rates when employing varying features.

Distance & Direction Feature Extraction: Let $\mathbf{x} = [x_1, x_2, \cdots, x_N]$ denote a sequence of N elements that can be *IKDs* or *IKRs*. We define its *spatial vector* as $\mathcal{S} = [y_1, y_2, \cdots, y_N]$. To construct $\mathcal{S}$, we first classify all N elements into M clusters. If x_i and x_j ($i, j \in \{1, 2, \cdots, N\}$) are similar, they are placed in the same cluster, with each cluster representing a distinguishable *IKD* or *IKR* group. We then sort the M clusters based on the value of a random element from each cluster, assigning cluster indices from 1 to M accordingly. We set $y_1 = C(x_i)$, where C is a function that returns the index of the cluster in which x_i is placed. By applying the spatial vector to both spatial distances and directions, we can extract their spatial distance and direction

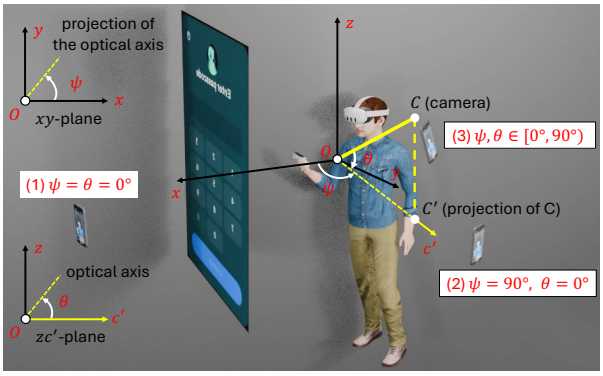


Fig. 5: Different recording scenarios.

features. In Appendix B, the algorithm for spatial vector generation is presented.

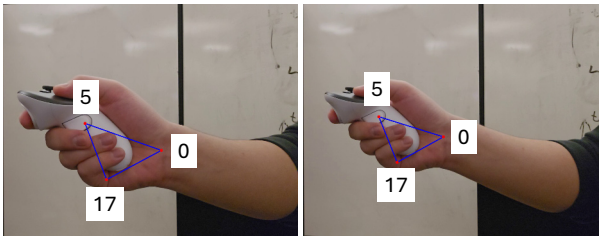
Position Transformation: From the video recordings, the attacker cannot directly access the virtual keypad, which is only visible to the user, and thus cannot get the *IKDs* during passcode input. Instead, a hand position transformation method needs to be developed to map the observed keypress locations to the corresponding positions on the virtual keypad. These mapped positions can then be utilized to calculate *IKDs*.

Empirically, the virtual keypad is designed to remain parallel to the user's body, ensuring ergonomic interaction. A coordinate system can be established, with the user's body aligning with the yz-plane and the xy-plane oriented perpendicular to it, as illustrated in Fig. 5. The x-axis represents the keypad's depth relative to the user. The camera's position is C, and its projection in the xy-plane is C'. We refer to the angle between the x-axis and the line OC' (referred to as the c'-axis) as the *yaw angle*, denoted as ψ, and the angle between the camera's optical axis (i.e., the direction that the camera faces) and the c'-axis as the *pitch angle*, denoted with θ. Particularly, we consider three typical scenarios according to the camera's location relative to the target user,

- *Frontal View:* record user input in front of the user from a central perspective, aligning the camera precisely with the middle of the virtual keypad, where $\psi = \theta = 0°$.
- *Side View:* the camera's optical axis is parallel to the y-axis, capturing user interactions with the virtual keypad from a lateral perspective, where $\psi = 90°$ and $\theta = 0°$.
- *General View:* the camera can face the user from an arbitrary direction in the first octant (i.e., the region where all three coordinates are positive), with $\psi, \theta \in [0°, 90°)$.

Note that ψ cannot exceed $90°$ until $270°$, as this would require recording keypresses from behind the user, where the user's body may obstruct the view of the controller or hand. Additionally, for ψ values between $0°$ and $-90°$, the camera is positioned on the opposite side of the $x-z$ plane, along the negative y-axis. Similarly, for θ values between $0°$ and $-90°$, the camera is positioned on the opposite side of the $x-y$ plane, along the negative z-axis. For distance feature extraction, these negative angles are equivalent to their positive counterparts.

Let N denote the PIN length. Two successive keypresses



(a) Near to the camera (b) Moving away from the camera

Fig. 6: Impact of recording depth: all segments (connecting keypoints 0 and 5, 5 and 17, and 17 and 0) decrease at the same ratio as the hand moves farther from the camera.

generate two keypress locations A_i and A_{i+1} ($i \in [1, N-1]$) in the video, corresponding to A'_i and A'_{i+1} in the virtual keypad. Using the measured distance $|A_iA_{i+1}|$, we compute the corresponding *IKD* on the virtual keypad, i.e., $|A'_iA'_{i+1}|$.

Frontal View Case: VR devices (e.g., Meta Quest 3) are often designed to mirror real-world hand movements in the virtual environment (i.e., applying a 1:1 mapping of virtual to physical space) to offer the highest level of immersion. Consequently, in the frontal view case, where the camera and the "laser pointer" movement on the virtual keypad are parallel, we have $|A'_iA'_{i+1}| = |A_iA_{i+1}|$.

Side View Case: In this scenario, the attacker's camera captures only the hand's movement along the z-axis directly, while the trajectory along the y-axis appears as a single projected point on the camera. Let d denote the distance variation for the two successive keypresses along the y-axis on the virtual keypad. We then have $|A'_iA'_{i+1}| = \sqrt{|A_iA_{i+1}|^2 + d^2}$. Now, the calculation of the moving distance on the virtual keypad (i.e., $|A'_iA'_{i+1}|$) can be simplified to computing the movement distance (i.e., d) along the y-axis on the virtual keypad.

Generally, when holding the controller, the user's grip position remains relatively fixed, maintaining a consistent orientation while adjusting the controller's position using their arm and wrist. We select two anchor points on the plane of the hand that is visible to the camera, while is parallel to the xz-plane. The segment connecting these two anchor points is denoted as S, and it is used as a reference for tracking the hand's relative position along the y-axis. When the hand moves closer to the camera, the segment S appears larger in the recording because objects closer to the camera occupy more pixels in the field of view. When the hand moves farther from the camera, the segment S appears smaller in the recording, as the relative size decreases with distance. We select 3 keypoints outputted by MediaPipe, indexed by 0, 5, and 17, as discussed in Section II-A). Fig. 6 illustrates how the segment lengths between keypoints change as the hand moves away from the camera. Let $S_{i,j}$ denote the measured length, in pixels, of the segment connecting keypoints i and j ($i, j \in \{0, 5, 17\}$). In Fig. 6a, we have $S_{0,5} = 545$, $S_{5,17} = 403$, and $S_{17,0} = 418$, while in Fig. 6b, these values decrease to 454, 331, and 354, denoted as $S'_{0,5}$, $S'_{5,17}$, and $S'_{17,0}$, respectively. We observe that the ratio remains consistent across all segments: $\frac{S_{0,5}}{S'_{0,5}} = \frac{S_{5,17}}{S'_{5,17}} = \frac{S_{17,0}}{S'_{17,0}} \approx 1.2$.

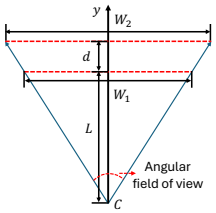


Fig. 7: Field of view of the camera for the side view case.

As shown in Fig. 7, when the hand moves for a distance of d along the y axis for two successive keypresses, a segment will be scaled down by the ratio $\frac{W_1}{W_2}$ in the camera's view. Let S_1 and S_2 ($S_2 \leq S1$) denote the observed reference segment widths in the recording video at the first and second keypresses (i.e., at distances L and $L+d$), respectively, where L denotes the distance between the camera and the hand at the first keypress. We thus have $S_2 = S_1 \cdot \frac{W_1}{W_2}$. According to the property of similar triangles regarding ratios, we have $\frac{W_1}{W_2} = \frac{L}{L+d}$. We can then compute d as $(S_1/S_2 - 1) \cdot L$. Similarly, if the hand moves along the reverse direction of the y axis, we have $S_2 > S1$ and $d = (S_2/S_1 - 1) \cdot L$. Consequently, we obtain

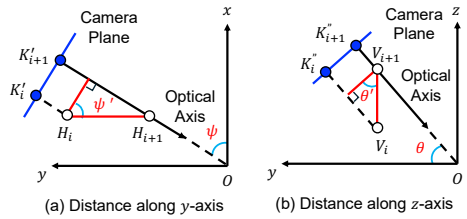
$$|A'_iA'_{i+1}| = \sqrt{|A_iA_{i+1}|^2 + (S_2/S_1 - 1)^2 \cdot L^2}. \quad (1)$$

General View Case: In a general case, the camera can be anywhere in the first octant. In this case, we convert the problem of calculating the moving distance on the virtual keypad (i.e., the yz-plane) into separately calculating the moving distances along y-axis and z-axis. Fig. 8 illustrates the relationship between movement distances along the y-axis and z-axis on the virtual keypad and their corresponding displacements in the camera's recording. Let $|H_iH_{i+1}|$ and $|V_iV_{i+1}|$ represent the movement distances along the y-axis and z-axis, respectively. The corresponding measured distances in the camera's recording are denoted as $|K'_iK'_{i+1}|$ and $|K''_iK''_{i+1}|$. With $\psi' = \psi$ and $\cos\psi' = \frac{|K'_iK'_{i+1}|}{|H_iH_{i+1}|}$, we have $|H_iH_{i+1}| = \frac{|K'_iK'_{i+1}|}{\cos\psi}$. Also, as $\theta' = \theta$ and $\cos\theta' = \frac{|K''_iK''_{i+1}|}{|V_iV_{i+1}|}$, we get $|V_iV_{i+1}| = \frac{|K''_iK''_{i+1}|}{\cos\theta}$. Consequently, the moving distance $|A'_iA'_{i+1}|$ on the virtual keypad can be computed as

$$\begin{aligned} |A'_iA'_{i+1}| &= \sqrt{|H_iH_{i+1}|^2 + |V_iV_{i+1}|^2} \\ &= \sqrt{\frac{|K'_iK'_{i+1}|^2}{\cos^2\psi} + \frac{|K''_iK''_{i+1}|^2}{\cos^2\theta}}. \end{aligned} \quad (2)$$

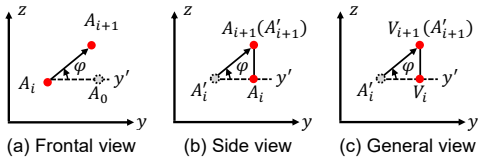
To obtain the *IKR* (denoted as φ) from the video recordings, we also consider the three aforementioned typical scenarios:

- *Frontal View:* The angle of laser point movement (from point A_i to A_{i+1}) can be directly observed. As shown in Fig. 9a, with $y' \parallel y$ and A_0 - a point on the line y', we have $\varphi = \angle A_{i+1}A_iA_0$.
- *Side View:* In Fig. 9b, we align the second keypress location, A_{i+1}, in the recorded video and A'_{i+1} in the virtual keypad. With the directly measured segment $|A_iA_{i+1}|$, and the computed *IKD* , $|A'_iA'_{i+1}|$, we obtain $\varphi = \arcsin\frac{|A_iA_{i+1}|}{|A'_iA'_{i+1}|}$.



(a) Distance along y-axis (b) Distance along z-axis

Fig. 8: The movement distance on the virtual keypad and its corresponding reflections on the recorded image.



(a) Frontal view (b) Side view (c) General view

Fig. 9: *IKR* in three different scenarios.

- *General View:* While computing *IKD*, we obtain the laser point movement distances along the y-axis and on the keypad, $|V_iV_{i+1}|$ and $|A'_iA'_{i+1}|$, respectively. In Fig. 9c, we align V_{i+1} with A'_{i+1}. Thus, we get $\varphi = \arcsin\frac{|V_iV_{i+1}|}{|A'_iA'_{i+1}|}$.

IKD Feature Extraction: For the Meta Quest 3 virtual passcode keypad, the horizontal center-to-center distance between adjacent keys in the same row is set as 1 unit. Consequently, the vertical center-to-center distance between adjacent keys in the same column is represented as 3/5 unit. Accordingly, all *IKDs* in the keypad form a set of $\{0, \frac{3}{5}, 1, \frac{6}{5}, \frac{\sqrt{34}}{5}, \frac{\sqrt{61}}{5}, \frac{9}{5}, 2, \frac{\sqrt{106}}{5}, \frac{\sqrt{109}}{5}, \frac{\sqrt{136}}{5}\}$. Some *IKDs* are quite close and the resultant hand movement distances may not show obvious difference, we divide all *IKDs* into the following five groups (g_1 to g_5): if two *IKDs* belong to the same group, they are categorized into the same subset, and vice versa.

- g_1: $IKD \in \{0\}$;
- g_2: $IKD \in \{\frac{3}{5}\}$;
- g_3: $IKD \in \{1, \frac{6}{5}, \frac{\sqrt{34}}{5}\}$;
- g_4: $IKD \in \{\frac{\sqrt{61}}{5}\}$;
- g_5: $IKD \in \{\frac{9}{5}, 2, \frac{\sqrt{106}}{5}, \frac{\sqrt{109}}{5}, \frac{\sqrt{136}}{5}\}$.

Based on the *IKDs* of a typed PIN, we can thus derive its spatial distance feature by computing the spatial vector. Similarly, for a 6-digit PIN, we can then obtain 541 subsets in total. Each subset has about $10^6/541 \approx 1,848$ passcodes. Thus, the corresponding feature distinction rate equals $541/10^6 = 5.41 \times 10^{-4}$.

IKR Feature Extraction: The *IKD* feature captures the displacement of the laser point between consecutive keystrokes but does not encode directional information, leading to ambiguity where multiple key pairs can have the same spatial distance. For example, the *IKD* values of key pairs "2-1" and "2-3" are identical. To address this issue, we consider eight common standard directions: east (0°), southeast (45°), south (90°), southwest (135°), west (180°), northwest (225°), and north (270°). When inputting passcodes, the laser point

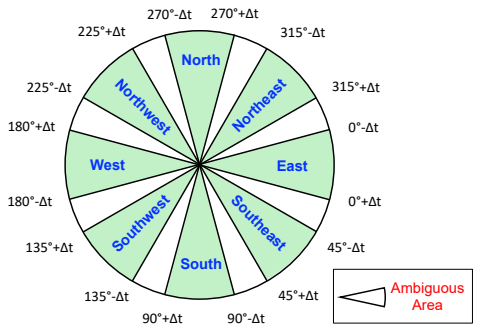


Fig. 10: Eight direction groups and ambiguous areas.

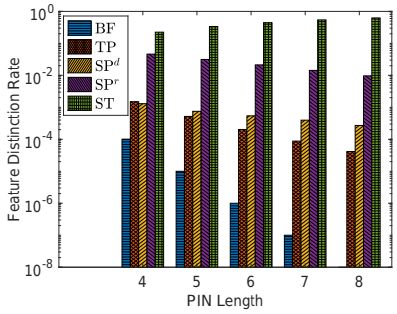


Fig. 11: Feature distinction rate comparison.

may not always be aimed precisely at the center of each key area. Therefore, we classify a movement as belonging to a standard direction if its direction falls within $\pm\Delta t$ degrees of that direction. This results in eight separately distinguishable direction regions. Note that $\Delta t \leq 22.5°$ to prevent overlap between two successive direction regions. When two *IKR* values fall within the same direction region, they are assigned the same directional label. As shown in Fig. 10, each pair of successive direction regions is separated by an ambiguous area with an angular width of $45° - 2\Delta t$. When the movement of the laser point falls within this ambiguous area, it can be categorized into either of its two neighboring direction regions.

In general, a smaller Δt increases the size of the ambiguous area, potentially leading to more candidates for a PIN whose *IKRs* lie within these areas. Meanwhile, it can tolerate larger *IKR* calculation errors (e.g., when the laser point does not perfectly align with the center of the key areas while pressing keys). In contrast, a larger Δt reduces the size of the ambiguous area, resulting in fewer PIN candidates overall. However, it tolerates smaller *IKR* calculation errors and may increase the probability of *IKR* feature misclassification. Empirically, we set $\Delta t = 14.5°$ to balance the feature classification and efficiency for PIN inference. With the *IKRs* of a typed PIN, we can obtain its spatial direction feature by computing the spatial vector. Accordingly, we get 21,211 groups to use *IKR* to divide all 6-digit PIN space, achieving a feature distinction rate of $21,211/10^6 = 0.021211$.

Feature Fusion & Correlation: Feature fusion combines the temporal and spatial (distance and direction) features into a spatiotemporal representation, which is more discriminative than any individual input feature. With a recording of typing an n-digit PIN, we can compute its temporal feature $\mathcal{T} = [r_1, r_2, \cdots, r_n]$, spatial distance feature $\mathcal{S}^d = [s_1^d, s_2^d, \cdots, s_{n-1}^d]$, and spatial direction feature $\mathcal{S}^r = [s_1^r, s_2^r, \cdots, s_{n-1}^r]$, enabling us to obtain its spatiotemporal feature $\mathcal{ST} = [r_1, s_1^d, s_1^r, r_2, s_2^d, s_2^r, \cdots, s_{n-1}^d, s_{n-1}^r, r_n]$. We utilize $\mathcal{ST}$ to divide all 6-digit PINs, and obtain 444,731 subsets in total, which is 2,190, 821, and 20 times more than those obtained using the temporal, spatial distance, or spatial direction feature, respectively. The resultant feature distinction rate becomes $444,731/10^6 = 0.444731$.

Fig. 11 illustrates feature distinction rates when searching for PIN candidates using traditional brute-force guessing (BF), the temporal feature (TP), the spatial distance feature (SP^d), the spatial direction feature (SP^r), and the spatiotemporal feature (ST), with the PIN length L varying from 4 to 8. An interesting observation emerges: as the PIN length increases, the feature distinction rates of BF, TP, SP^d, and SP^r all decrease, indicating that the PIN inference becomes more challenging due to a larger average number of PINs per subset. In contrast, the feature distinction rate for ST gradually increases. This finding demonstrates that the spatiotemporal feature consistently helps narrow the search space for the typed PIN and becomes even more effective for longer PINs.

V. Experimental Evaluation

We develop an Android app to implement *MotionDecipher* and test it on off-the-shelf smartphones, targeting to infer PIN input on popular VR devices. The prototype system consists of a VR user (victim) and an attacker using a common smartphone (i.e., Samsung S23 [42]) to record the user's hand-controller interactions while inputting PINs on a VR device. The smartphone is configured with standard settings and records 1080p videos at 60 FPS. The inputted PINs are extracted from the RockYou 2024 dataset [43], a well-known repository of real-world leaked PINs. The recorded video is then processed by the app of *MotionDecipher* to generate a candidate list of the target PIN entered by the victim.

Metrics: We use the following two metrics.

- *Entropy:* It measures the PIN strength against brute-force attacks. Suppose there are m candidates for a PIN X and x_i ($i \in \{1, 2, \cdots, m\}$) denotes one of them. The X's entropy $H(X)$ equals $-\sum_{i=1}^{m} P(x_i) \cdot \log_2 P(x_i)$, where $P(x_i)$ is the probability that $X = x_i$ holds.
- *Top-k Accuracy:* It represents the probability that the final candidate list includes the target PIN inputted by the victim, denoted by α. With m final candidates, we have $\alpha = k/m$ if k is smaller than m, otherwise, $\alpha = 1$.

A. Case Study

In this example, the user enters a six-digit PIN - "260422" to unlock a Meta Quest 3. We record the user's controller operation from a distance larger than three meters. Using the recorded video, we successfully identified all keypress events.

Fig. 12 presents six extracted individual keypress frames alongside their detected keypress locations on the virtual

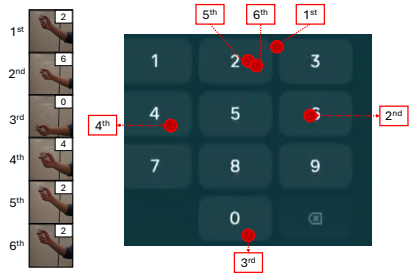


Fig. 12: Raw keypress frames and the keypress locations on the virtual keypad.

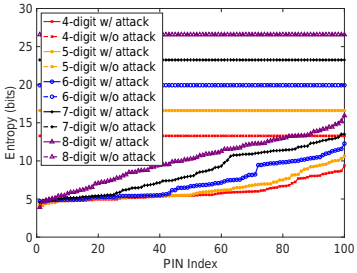


Fig. 13: Entropy distribution for PINs with varying lengths.

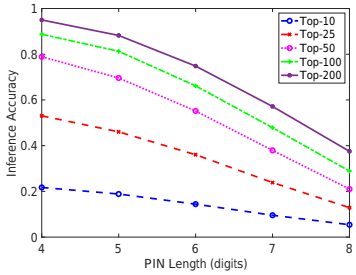


Fig. 14: Average top-k accuracy for inferring PINs with varying lengths.

keypad. As observed, actual keypresses do not always align perfectly with the center of key areas, highlighting the necessity of our motion demodulation algorithm, which effectively tolerates off-center keypresses. The temporal location repetition feature is $[1, 2, 3, 4, 1, 1]$; the spatial distance feature is $[2, 3, 3, 2, 1]$; the spatial direction feature is [east or southeast, west or southwest, northwest, east or northeast, east]. With feature fusion and correlation, *MotionDecipher* yields three candidate PINs - "238122", "260422", and "560455". We see that the second candidate is the correct one. Thus, *MotionDecipher* substantially reduces the maximum attempts for breaking this 6-digit PIN to just 3, compared with the brute force attack that needs 10^6 times. Accordingly, the PIN entropy is decreased from $6 \log_2 10 = 19.93$ bits to $\log_2 3 = 1.58$ bits, and the top-k accuracy equals 100% when $k \geq 3$.

B. Inference of PINs with Varying Lengths

PINs are commonly four digits for ease of memorization and usability [44], and 6-digit passcodes are also popular [45], [46]. For security considerations, the PIN length nowadays can be diversified. To approximate user choices of passcodes in practice, we extract leaked real-world PINs with 4 to 8 digits from the *RockYou 2024* database. For every passcode length, we obtain 100 samples, and type each extracted passcode separately when using a Meta Quest 3. The attacker uses a Samsung S23 and launched *MotionDecipher*, computing passcode entropies and top-k accuracies.

We sort the PINs of each length in ascending order of the entropies and index them from 1 to 100 in increments of 1, as shown in Fig. 13. For comparison, we also show the PIN entropy distribution without applying the proposed attack. We observe three major trends. First, with *MotionDecipher*, the search space of the typed passcode with different lengths is significantly shrunk. The attack decreases the entropy of an 8-digit PIN from 26.6 bits to as low as 3.9 bits, vastly reducing the maximum brute-force attempts for breaking the passcode from 100 million to just 15. Second, PINs of the same length exhibit significant variations in entropy, indicating notable differences in security levels, and longer PINs may introduce greater security disparities. Third, longer PINs do not necessarily enhance security and may even weaken it, as indicated by similar entropy distributions across different PIN lengths. This counterintuitive result occurs because longer PINs offer attackers richer spatiotemporal features, enabling them to narrow the search space more effectively. For example, *MotionDecipher* reduces entropy by an average of 7.4, 10.3, 12.7, 14.7, and 16.5 bits for 4- to 8-digit PINs, respectively. Consequently, cracking a 6-, 7-, or 8-digit PIN may be simpler than brute-forcing a 3-digit PIN, and meanwhile, deducing a 4- or 5-digit PIN may require less effort than brute-forcing a 2-digit PIN. In addition, we count the number of guesses required to infer each selected PIN. With *MotionDecipher*, 78% of the chosen 4-digit PINs can be inferred within 100 guesses, 77% of 5-digit PINs within 150 guesses, and 70% of 6-digit PINs within 300 guesses. For longer PINs, 60% of 7-digit PINs can be inferred within 800 guesses, and 52% of 8-digit PINs within 1,500 guesses.

Fig. 14 illustrates the average top-k accuracy for varying PIN lengths. We see that *MotionDecipher* can consistently achieve high inference performance for all PIN lengths, and the mean top-k accuracy exhibits a decline as the key length increases. Particularly, the mean top-100 accuracy ranges from 28.9% to 88.8% across PIN lengths, demonstrating that up to 100 guesses are sufficient to infer a substantial portion of PINs, depending on their length. Also, our attack achieves at least 13% mean top-25 accuracy across all PIN lengths.

C. Robustness to Influential Factors

The recording distance/angle/device may vary for the attacker, and the victim may use different VR devices. In the following, we evaluate the impact of those factors. We randomly select 100 4-digit PINs from the RockYou 2024 database, and ask the user to input each selected PIN wearing a Meta Quest 3, under each situation.

1) Impact of Recording Distances: While the user is immersed in the VR environment, an attacker's close physical proximity may alert the user to their presence [47], especially since some VR headsets feature capabilities like Passthrough on Meta Quest headsets [48], which allow users to observe their physical world surroundings through the headsets' integrated cameras. To avoid suspicion, we vary the recording distance from 3 to 8 meters in increments of 1 meter.

Fig. 15 illustrates the entropy distribution for varying recording distances. We see that, regardless of the recording distance, the search space of the typed PIN is significantly

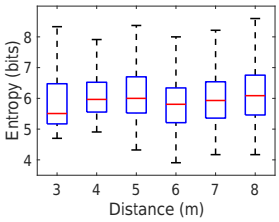


Fig. 15: Entropy distribution vs. distance.

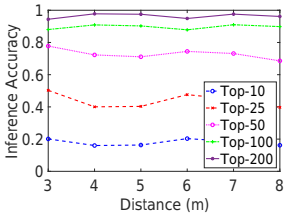


Fig. 16: Average top-k accuracy vs. distance.

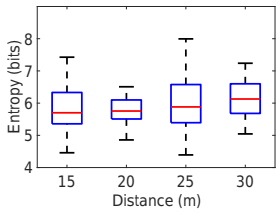


Fig. 17: Entropy distribution for long distances.

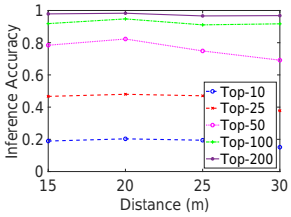


Fig. 18: Average top-k accuracy for long distances.

reduced. Overall, the attacker decreases the entropy of a 4-digit PIN from 13.2 bits to a range between 3.9 to 8.5 bits. This vastly lowers the maximum brute-force attempts required to break the PIN, reducing them from 10,000 to as few as 15. Particularly, at each recording distance from 3 to 8 meters, at least 54% of the selected PINs can be recovered in fewer than 70 guesses. Also, the entropy ranges for different recording distances are quite similar. Fig. 16 presents the corresponding inference accuracy. We observe that our attack consistently reduces PIN strength across all recording lengths. The top-10 and top-25 accuracies range from 16.2% to 20.3% and 40.0% to 50.3%, respectively, while the top-50, top-100, and top-200 accuracies remain above 69.0%, 87.9%, and 94.8%, respectively.

Long Recording Distance Tests: Digital zoom is a software-based feature commonly used in smartphones. Unlike optical zoom, which physically adjusts the lens to magnify a subject, digital zoom works by enlarging a specific portion of an image. This process may result in a significant loss of image quality, as no additional information is added to the magnified image. Some of the latest smartphones, though not all, are equipped with optical zoom lenses. These lenses can be adjusted to bring the target closer or farther away while preserving the clarity and detail of the image. We employ a Samsung S23 Ultra smartphone with 10 times optical zoom [49] and a 36× budget telephone lens (40 US dollars around) [50] to perform the experiments. Our long-distance tests are conducted in a long corridor of a building. We vary the recording distance from 15 to 30 m in increments of 5 m.

Figures 17 and 18 present the resultant entropy distribution and top-k accuracy. We see that, overall, for the tested long distances, the inference performance of *MotionDecipher* remains consistent with cases where the recording distance is between

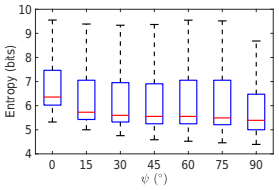


Fig. 19: Entropy vs. ψ.

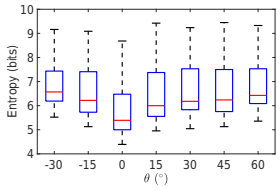


Fig. 20: Entropy vs. θ.

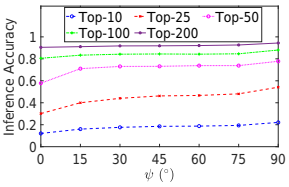


Fig. 21: Accuracy vs. ψ.

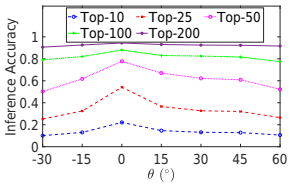


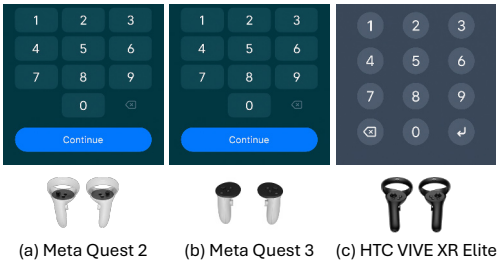
Fig. 22: Accuracy vs. θ.

3 and 8 meters. Specifically, the average entropy for varying long distances ranges from 5.8 to 6.2 bits; the average top-100 and top-200 accuracies consistently exceed 92% and 97%, respectively, across all cases. Also, *MotionDecipher* infers at least 50% of PINs within 70 guesses at each tested long distance, showing its practicality for long-range attacks.

2) Impact of Recording Angles: The attacker may record the typing from various directions, characterized by yaw and pitch angles (i.e., ψ and θ), as defined in Section IV-C1. We maintain θ fixed and vary ψ from $0°$ to $90°$, with increments of $15°$. Next, with a fixed ψ, we vary θ from $-30°$ to $60°$, with increments of $15°$. Figures 19 and 20 present the obtained entropy distribution. We see that with a fixed θ, the average entropy ranges from 5.9 to 6.7 bits; with a fixed ψ, the average entropy ranges from 5.9 to 6.9 bits. Meanwhile, as ψ increases, the average entropy gradually decreases. This appears to be the case because a larger yaw angle generally offers a clearer view of both the controller's trigger and the user's finger pressing it, thereby improving the accuracy of keypress detection. Similarly, as θ increases from $0°$ to $60°$, or shifts from $0°$ to $-30°$, the average entropy slightly increases. By counting the number of inferred candidates for each selected PIN, we see that 66-76% of PINs across yaw angles and 56-76% across pitch angles can be inferred within 100 guesses.

Figures 21 and 22 show the corresponding average top-k accuracy. We see that *MotionDecipher* reduces the PIN strength across all test yaw or pitch angles. Particularly, with a fixed θ, the top-100 accuracy for all ψ consistently exceeds 0.81. Like the entropy variation, the top-k accuracy slightly increases with ψ. Also, with a fixed ψ, the average top-k accuracy exhibits a rise-and-fall pattern, peaking at $\theta = 0°$, where the minimum average PIN entropy occurs. In addition, Appendix C and Appendix D present the impact of recording frame rates and resolutions, respectively.

3) Impact of VR Devices: We test three popular VR devices: Meta Quest 2, Meta Quest 3, and HTC VIVE XR Elite, abbreviated as Quest 2, Quest 3, and VIVE XR, respectively. As shown in Fig. 23, Quest 2 and Quest 3 share the same



(a) Meta Quest 2 (b) Meta Quest 3 (c) HTC VIVE XR Elite

Fig. 23: Virtual PIN pad layouts and corresponding controllers.

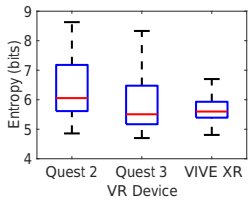


Fig. 24: Entropy vs. VR device.

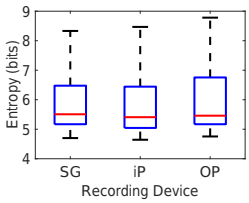


Fig. 25: Entropy vs. recording device.

virtual PIN pad layout, whereas VIVE XR adopts a similar design; the controllers of both Quest 2 and VIVE XR feature tracking rings encircling the top, while Quest 3 controllers have no visible tracking ring. Fig. 24 presents the observed entropy distributions. All devices exhibit similar median entropy values (Quest 2: 6.0 bits; Quest 3: 5.5 bits; VIVE XR: 5.6 bits). In terms of average entropy, the VIVE XR slightly outperforms the others, with Quest 2, Quest 3, and VIVE XR achieving 6.4, 6.0, and 5.8 bits, respectively. This improvement may stem from the smaller key sizes on the VIVE XR's virtual PIN pad. Smaller keys constrain the laser pointer's movement when selecting digits, resulting in more distinguishable inter-key distances, and consequently, more informative spatiotemporal features. Across different VR headsets, *MotionDecipher* infers 66% of the selected PINs within 100 guesses on Meta Quest 2, 76% on Meta Quest 3, and 88% on HTC VIVE, demonstrating robust performance across platforms.

Table II shows the corresponding average top-k accuracy. We see that all three VR devices can achieve no smaller than 0.67 top-50 accuracy. Similarly, due to the smaller key sizes, VIVE XR has slightly higher top-k accuracy when $k \in \{50, 100, 200\}$, accordingly. These results convincingly demonstrate the effectiveness of *MotionDecipher* against diverse VR devices operated via handheld controllers.

4) Impact of Recording Devices: We experiment with cameras on three popular smartphones, Samsung Galaxy S23, iPhone 14, and OnePlus 12, referred to as SG, iP, and OP, respectively. Fig. 25 presents the corresponding entropy distribution. We see that for all recording devices, *MotionDecipher* makes breaking PINs much easier than traditional brute-force attacks. The average entropies for SG, iP, and OP are 5.99, 5.93, and 6.05 bits, respectively, with no significant variation. Across smartphones, *MotionDecipher* infers 72% of PINs on OP, 76% on SG, and 80% on iP within 100 guesses, demonstrating its reliability on consumer-grade recording devices.

TABLE II: Average top-k accuracy vs. VR device.

Device	Top-10	Top-25	Top-50	Top-100	Top-200
Quest 2	0.15	0.37	0.67	0.83	0.94
Quest 3	0.20	0.50	0.78	0.88	0.94
VIVE XR	0.16	0.40	0.71	0.86	0.93

TABLE III: Average top-k accuracy vs. recording device.

Device	Top-10	Top-25	Top-50	Top-100	Top-200
OnePlus 12	0.19	0.48	0.74	0.87	0.93
Samsung S23	0.20	0.50	0.78	0.88	0.94
iPhone 13	0.21	0.53	0.78	0.88	0.94

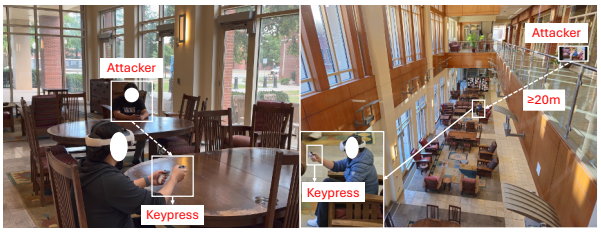
Table III presents the corresponding mean top-k accuracy. We observe that all recording devices achieve consistent inference performance; in particular, the top-100 accuracy equals or exceeds 0.87. These results suggest that *MotionDecipher* is robust against different recording devices.

VI. Real-world User Study

We recruited 10 volunteers (U1-U10; aged 21-36 years old; 4 females and 6 males), all of whom are active VR users, to examine the practicality of *MotionDecipher*.[1] Experiments were conducted in a lobby, where each participant was instructed to sit on a chair, put on a Meta Quest 3 headset, and proceed to unlock it. Benefiting from the ergonomic design of the controllers, all participants naturally chose to use them for PIN input. We consider two typical scenarios based on the distance between the attacker and the victim, as shown in Fig. 26: (a) *normal-distance: MotionDecipher* runs on a Samsung S23 smartphone placed at a distance greater than 3 meters but less than 20 meters from the victim; (b) *long-distance: MotionDecipher* runs on a Samsung S23 Ultra smartphone equipped with 10× optical zoom, positioned more than 20 meters away from the victim. Each participant freely adjusted their sitting positions and randomly selected a 6-digit PIN as the target PIN. We allow the participants enough time to memorize and practice their selected PINs before testing. Also, we reminded participants not to choose their true in-use PINs for their devices or applications. For each scenario, every participant performed 50 attempts with different PINs. We present the inference results to the corresponding participant, who determines whether the inferred PIN candidate list contains the input PIN. For all trials of all participants, we find that the input PIN is always included in the inference result. Also, 70% of the PINs selected by all users can be inferred within 300 guesses under the normal-distance scenario, and 60% under the long-distance scenario.

Figures 27 and 28 show the resultant entropy distributions across all users. We see that in both scenarios, *MotionDecipher* consistently reduces PIN strength across all users. Also, overall, the entropies in the normal-distance scenario are slightly lower than those in the long-distance scenario. Specifically, in the normal-distance scenario, the mean entropy across users ranges from 6.68 to 8.20 bits, whereas in the long-distance scenario, it ranges from 7.35 to 8.73 bits. This appears as shorter recording distances lead to more accurate keypress

[1]The study has been reviewed and approved by our institution's IRB.



(a) Normal-distance Scenario (b) Long-distance Scenario

Fig. 26: Real-world scenarios.

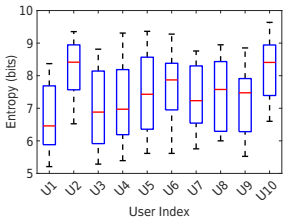


Fig. 27: Entropy in the normal-distance scenario.

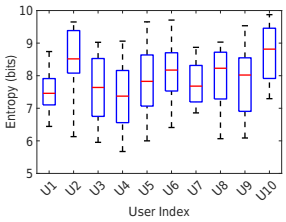


Fig. 28: Entropy in the long-distance scenario.

event detection and spatiotemporal feature extraction. Also, the long-distance scenario involves a more complex recording environment, introducing additional occlusions that hinder accurate hand movement tracking. Figures 29 and 30 illustrate the corresponding average top-k accuracy values. We observe in the normal-distance scenario, the top-200 accuracy ranges from 0.64 to 0.95, while it varies from 0.53 to 0.89 in the long-distance scenario. This indicates in both scenarios, over half of the selected 6-digit PINs can be successfully inferred within 200 guesses for all users.

VII. Countermeasures

A. Mitigating PIN Guessing via Backoff and Lockout

To defend against brute-force attacks against PIN authentication, real-world systems often implement exponential backoff or account lockout mechanisms to avoid excessive PIN guesses in a limited time. For instance, iOS Lock Screen imposes delays of 1 min, 5 min, 15 min, 1 hr, 3 hrs, and 8 hrs after 4 to 9 failed passcode attempts, disabling the device after 10 [51]. VR systems, including Meta Quest 2 and 3, as discussed in Section I, currently employ a tiered PIN retry strategy that combines fixed and slowly escalating delays, prioritizing usability by allowing dozens of guesses without enforcing prohibitive wait times. Such a weak policy would make the systems more vulnerable to the proposed attack.

On the other hand, for a system equipped with the defense of exponential backoff or account lockout, the effective success rate of *MotionDecipher* depends on the number of allowed PIN attempts (denoted as L_{max}) relative to the inferred candidate list size (denoted as K). If $L_{max} \geq K$, the attacker can test all candidates, and the success rate remains at the original top-K accuracy. However, if $L_{max} < K$, the success rate decreases proportionally to L_{max}/K. Meanwhile, when L_{max} is set too small, it may significantly degrade usability, as legitimate users may be locked out after a few mistakes.

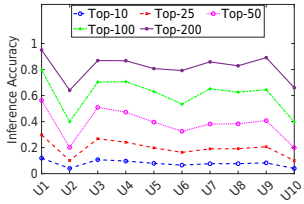


Fig. 29: Top-k accuracy in the normal-distance scenario.

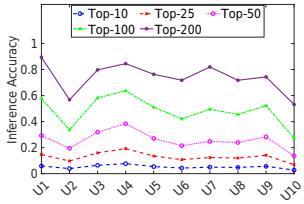


Fig. 30: Top-k accuracy in the long-distance scenario.

B. Deploying External Barriers

An intuitive defense is to use physical barriers that obstruct an attacker's line-of-sight (LoS) to the VR user's controllers during PIN entry. However, as *MotionDecipher* works across a broad range of angles, this approach would require pre-deploying additional physical hardware (i.e., a large barrier), making it impractical and non-portable. Moreover, introducing physical objects into the user's VR environment (i.e., the physical space around them) may restrict user movement and even pose safety risks, ultimately undermining the goal of an immersive and seamless VR experience.

C. Randomizing Keypad Layouts

A viable defense is to randomize the keypad layout for each digit input, such that the disclosed spatiotemporal features are obfuscated. Specifically, repeated keypress locations do not necessarily correspond to the same digit, and the inter-key distance or direction between a pair of digits is no longer fixed. This method introduces ambiguity into motion patterns without requiring external barriers. However, such a software-based method requires changes at the level of User Interface (UI) to enable the random rearrangement of keys after each digit entry, which may slow down the overall PIN entry process. Also, User Experience (UX) will be affected, as users may find it challenging to adapt to this dynamic input method.

D. Adding Guard Digits

Another software-based defense is to type specifically generated extra digits visible to the VR user while unknown to the attacker, referred to as *guard digits*. Since inserting digits among the actual PIN sequence may disrupt the input process, we position guard digits at the beginning, the end, or both of the actual PIN. Such extra input would force the attacker to determine which part of the inferred digit sequence corresponds to the actual PIN. Our work reveals that a longer VR PIN does not necessarily have higher security, as discussed in Section V-B. Thus, the guard digits need to be carefully designed. Let N_0 and N_g denote the number of candidate PINs without and with guard digits, respectively. If $N_g > N_0$, this strategy increases the attack difficulty. Suppose the user types L guard digits and the actual PIN, which can appear before the first guard digit, between any two consecutive guard digits, or after the last guard digit. There are $L+1$ possible positions for the actual PIN within each candidate of the typed whole

digit sequence. Consequently, considering possible repetition, the attacker would have up to $N_g(L+1)$ PIN candidates.

Compared with the method of randomizing keypad layouts, which typically requires $(N-1)$ keypad rearrangements for entering an N-digit PIN, this approach also necessitates UI-level modifications but generally requires only a single keypad change per PIN entry to accommodate the input of guard digits. From this perspective, this method is easier to implement than the defense of randomizing keypad layouts. Besides, it involves entering additional digits, which introduces UX changes and may lead to a longer PIN input time.

VIII. Related Work

A. Traditional Keystroke Inference

Extensive research has been aimed at inferring keystrokes in non-VR environments, particularly through non-invasive approaches, which mainly include the following categories.

Vision-based: An attacker can stealthily record the typing process. In traditional settings, where users type directly on physical or touchscreen keyboards with their fingers, attackers can recover keystrokes by tracking keypress-induced movements of the hand or fingertips [10], [16], [52], [53], upper body [12], tablet backside [15], or even eye gaze [13]. Also, typed input can be revealed by exploiting key pop-out events [14] or analyzing hand motion [54] in reflections on surfaces such as sunglasses. However, due to the nature of VR, both the virtual PIN pad and the user's eyes are enclosed within a headset, effectively eliminating the possibility of visual leakage via reflections or eye movement. Furthermore, users do not directly interact with a physical or touchscreen keyboard but input in the air using handheld controllers, resulting in unique motions that differ significantly from those observed in traditional keystroke scenarios. Consequently, traditional vision-based keystroke inference techniques are rendered ineffective in VR environments.

Sensor-based: Data captured via diverse on-board sensors, such as microphones [55], [56], [57], [58] and Inertial Measurement Unit (IMU) sensors (e.g., accelerometers and gyroscopes) [59], [60], [61], [62], can also be utilized to infer keystrokes. Such techniques, however, require placing the spying sensor in close proximity to the target.

Wireless-based: Wireless signals have shown success in inferring sensitive information related to human motion [63]. By placing a wireless receiver near the target keyboard in a wireless environment, an attacker can capture keystroke-disturbed wireless signals to infer keystrokes [64], [65], [66], [67], [68]. However, wireless-based techniques are often sensitive to environmental motion.

B. VR Keystroke Inference

Invasive: Attackers are assumed to have the capability to pre-install malware on the victim's VR devices, which stealthily transmits collected sensitive data, such as hand motion [6], [7], head orientation and location information [8], or headset accelerometer and gyroscope readings [9], back to attackers. However, such invasive attacks can be mitigated by anti-malware techniques, and may be infeasible for cautious VR users, who verify the legitimacy of each installed app.

Non-invasive: In this category, similar to traditional keystroke inference techniques, an attacker records the VR typing process using a camera, or places an on-board sensor or wireless receiver near the victim to collect keystroke-correlated data. However, existing video-based VR keystroke inference techniques (e.g., [6], [7], [69]) only work for input with hand gestures, and cannot work for input with controllers. Controllers are physically large objects that may block parts of users' hands from attackers' cameras. Also, users' fingers are wrapped around the controllers, and such an unnatural posture makes it difficult to distinguish finger movements.

Regarding sensor-based attacks, since VR keystrokes occur in mid-air rather than on a physical surface, they would not generate vibration signals that could be detected by IMU sensors near the victim. Also, due to the rapid attenuation of acoustic signals with distance, the attacker's microphone must be placed in close proximity to the victim (e.g., within 2.2 meters [18]), which greatly limits the practicality of acoustic-based techniques. In addition, wireless-based methods (e.g., [17]) often require deliberate placement of both a transmitter and receiver, with the victim positioned between them, and their performance can be adversely affected by environmental changes. Given that the VR PIN input process involves the coordinated motion of users' heads, bodies, hands, and controllers, such intertwined movement poses significant challenges for wireless-based methods. Moreover, these methods typically require user-specific training, as gesture patterns vary in duration and motion across individuals.

IX. Conclusion

This paper presents *MotionDecipher*, a novel video-based attack that infers PINs entered in VR. *MotionDecipher* operates under a realistic threat model, assuming only visual access to users' hand-controller interaction during PIN entry. It builds spatiotemporal correlations between the typed PIN and the corresponding recorded video footage. Compared with previous VR keystroke inference techniques, *MotionDecipher* has the following advantages: (1) non-invasive, it does not require pre-infecting the target VR headset with malware; (2) no close proximity required, it works from distances exceeding 20 meters; (3) effective for controller-based input, it is the first work that supports input via controllers; (4) no user-specific training, it works for new users whose data were not used in training the keypress event classifier. Experimental results demonstrate that *MotionDecipher* can significantly reduce PIN strength in various settings and environments, raising the immediate need for VR users to protect their typing privacy.

Acknowledgements

We would like to thank all anonymous reviewers for their insightful comments. This work was supported in part by the National Science Foundation under Grants No. 2155181 and No. 2424439.

REFERENCES

[1] Polaris Market Research & Consulting, Inc., "Virtual reality market share, size, trends, industry analysis report," https://www.polarismarketresearch.com/industry-analysis/virtual-reality-market, 2024.

[2] "Us virtual reality market size," https://www.oberlo.com/statistics/us-virtual-reality-market-size, 2024.

[3] Meta, "What is the metaverse?" https://about.meta.com/what-is-the-metaverse/, 2025.

[4] ——, "Manage your passcode on meta quest," https://www.meta.com/help/quest/articles/accounts/account-settings-and-management/passcode-meta-quest/, 2025.

[5] HTC Corporation, "Vive focus 3 support - settings: Setting a device passcode," https://www.vive.com/us/support/focus3/category_howto/setting-device-lock.html, 2025.

[6] 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)*, 2019, pp. 241–249.

[7] 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)*, 2022, pp. 765–774.

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

[9] 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)*. Anaheim, CA: USENIX Association, Aug. 2023, pp. 159–174.

[10] D. Balzarotti, M. Cova, and G. Vigna, "Clearshot: Eavesdropping on keyboard input from video," in *2008 IEEE Symposium on Security and Privacy (SP 2008)*, 2008, pp. 170–183.

[11] D. Shukla, R. Kumar, A. Serwadda, and V. V. Phoha, "Beware, your hands reveal your secrets!" in *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '14. New York, NY, USA: Association for Computing Machinery, 2014, p. 904–917.

[12] M. Sabra, A. Maiti, and M. Jadliwala, "Zoom on the keystrokes: Exploiting video calls for keystroke inference attacks," in *Network and Distributed Systems Security (NDSS) Symposium 2021*, 2021.

[13] 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)*, 2018, pp. 144–160.

[14] R. Raguram, A. M. White, D. Goswami, F. Monrose, and J.-M. Frahm, "ispy: automatic reconstruction of typed input from compromising reflections," in *Proceedings of the 18th ACM Conference on Computer and Communications Security*, ser. CCS '11. New York, NY, USA: Association for Computing Machinery, 2011, p. 527–536.

[15] J. Sun, X. Jin, Y. Chen, J. Zhang, Y. Zhang, and R. Zhang, "Visible: Video-assisted keystroke inference from tablet backside motion," in *Network and Distributed System Security Symposium*, 2016.

[16] Q. Yue, Z. Ling, X. Fu, B. Liu, K. Ren, and W. Zhao, "Blind recognition of touched keys on mobile devices," in *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '14. New York, NY, USA: Association for Computing Machinery, 2014, p. 1403–1414.

[17] 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)*, 2021, pp. 564–572.

[18] 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.

[19] W. Mao, M. Wang, W. Sun, L. Qiu, S. Pradhan, and Y.-C. Chen, "Rnn-based room scale hand motion tracking," in *The 25th Annual International Conference on Mobile Computing and Networking*, ser. MobiCom '19. New York, NY, USA: Association for Computing Machinery, 2019.

[20] D. Li, S. Cao, S. I. Lee, and J. Xiong, "Experience: practical problems for acoustic sensing," in *Proceedings of the 28th Annual International Conference on Mobile Computing And Networking*, ser. MobiCom '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 381–390.

[21] "MediaPipe," https://github.com/google-ai-edge/mediapipe, 2025.

[22] Google AI, "Hand landmarks detection guide for python," https://ai.google.dev/edge/mediapipe/solutions/vision/hand_landmarker/python, 2025.

[23] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2014, pp. 580–587.

[24] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," *IEEE transactions on pattern analysis and machine intelligence*, vol. 39, no. 6, pp. 1137–1149, 2016.

[25] K. He, G. Gkioxari, P. Dollár, and R. Girshick, "Mask r-cnn," in *Proceedings of the IEEE international conference on computer vision*, 2017, pp. 2961–2969.

[26] J. Donahue, L. A. Hendricks, M. Rohrbach, S. Venugopalan, S. Guadarrama, K. Saenko, and T. Darrell, "Long-term recurrent convolutional networks for visual recognition and description," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 39, no. 4, pp. 677–691, 2017.

[27] K. Xu, J. Ba, R. Kiros, K. Cho, A. C. Courville, R. Salakhutdinov, R. S. Zemel, and Y. Bengio, "Show, attend and tell: Neural image caption generation with visual attention," *arXiv preprint arXiv:1502.03044*, 2015.

[28] K. Cho, B. Van Merriënboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using rnn encoder-decoder for statistical machine translation," *arXiv preprint arXiv:1406.1078*, 2014.

[29] Y. LeCun, B. Boser, J. S. Denker, D. Henderson, R. E. Howard, W. Hubbard, and L. D. Jackel, "Backpropagation applied to handwritten zip code recognition," *Neural Computation*, vol. 1, no. 4, pp. 541–551, 1989.

[30] Y. Lecun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[31] S. Hochreiter, "Long short-term memory," *Neural Computation MIT-Press*, 1997.

[32] Meta, "Opt in or out of a Meta Quest pin for purchases," https://www.meta.com/help/quest/articles/accounts/account-settings-and-management/opt-in-or-out-meta-quest-pin/, 2024.

[33] A. Kirillov, E. Mintun, N. Ravi, H. Mao, C. Rolland, L. Gustafson, T. Xiao, S. Whitehead, A. C. Berg, W.-Y. Lo, P. Dollár, and R. Girshick, "Segment anything," *arXiv:2304.02643*, 2023.

[34] "Opencv: Opencv modules," https://docs.opencv.org/4.x, 2024.

[35] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[36] L. Bottou, "Large-scale machine learning with stochastic gradient descent," in *Proceedings of COMPSTAT'2010: 19th International Conference on Computational StatisticsParis France, Keynote, Invited and Contributed Papers*. Springer, 2010, pp. 177–186.

[37] "tf.keras.layers.randomflip," https://www.tensorflow.org/api_docs/python/tf/keras/layers/RandomFlip, 2024.

[38] "tf.keras.layers.randomrotation," https://www.tensorflow.org/api_docs/python/tf/keras/layers/RandomRotation, 2024.

[39] Y. Bengio, "Practical recommendations for gradient-based training of deep architectures," in *Neural networks: Tricks of the trade: Second edition*. Springer, 2012, pp. 437–478.

[40] L. Prechelt, "Early stopping-but when?" in *Neural Networks: Tricks of the trade*. Springer, 2002, pp. 55–69.

[41] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," 2017. [Online]. Available: https://arxiv.org/abs/1412.6980

[42] Samsung, "Galaxy S23," https://www.samsung.com/us/smartphones/galaxy-s23/, 2024.

[43] "Rockyou2024 : The largest password compilation leak," https://github.com/exploit-development/RockYou2024, 2024.

[44] E. von Zezschwitz, P. Dunphy, and A. De Luca, "Patterns in the wild: a field study of the usability of pattern and pin-based authentication on mobile devices," in *Proceedings of the 15th International Conference on Human-Computer Interaction with Mobile Devices and Services*, ser. MobileHCI '13. New York, NY, USA: Association for Computing Machinery, 2013, p. 261–270.

[45] D. Wang, Q. Gu, X. Huang, and P. Wang, "Understanding human-chosen pins: Characteristics, distribution and security," in *Proceedings of*

the 2017 ACM on Asia Conference on Computer and Communications Security, ser. ASIA CCS '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 372–385.

[46] J. H. Huh, H. Kim, R. B. Bobba, M. N. Bashir, and K. Beznosov, "On the memorability of system-generated PINs: Can chunking help?" in *Eleventh Symposium On Usable Privacy and Security (SOUPS 2015)*. Ottawa: USENIX Association, Jul. 2015, pp. 197–209. [Online]. Available: https://www.usenix.org/conference/soups2015/proceedings/presentation/huh

[47] G. Ye, Z. Tang, D. Fang, X. Chen, K. I. Kim, B. Taylor, and Z. Wang, "Cracking android pattern lock in five attempts," in *Proceedings of the 2017 Network and Distributed System Security Symposium 2017 (NDSS 17)*. Internet Society, 2017.

[48] Meta, "How to use Passthrough on Meta Quest," https://www.meta.com/help/quest/articles/getting-started/getting-started-with-quest-pro/full-color-passthrough/, 2025.

[49] Samsung, "Galaxy s23 ultra," https://www.samsung.com/us/smartphones/galaxy-s23-ultra/, 2025.

[50] "Amazon product - cellphone camera lens," https://www.amazon.com/dp/B0DJ4JR82N, 2025.

[51] Apple Inc., "Apple platform security," https://support.apple.com/en/guide/security/sec20230a10d/web, 2025.

[52] Q. Yue, Z. Ling, W. Yu, B. Liu, and X. Fu, "Blind recognition of text input on mobile devices via natural language processing," in *Proceedings of the 2015 Workshop on Privacy-Aware Mobile Computing*, ser. PAMCO '15. New York, NY, USA: Association for Computing Machinery, 2015, p. 19–24. [Online]. Available: https://doi.org/10.1145/2757302.2757304

[53] Z. Yang, Y. Chen, Z. Sarwar, H. Schwartz, B. Y. Zhao, and H. Zheng, "Towards a general video-based keystroke inference attack," in *32nd USENIX Security Symposium (USENIX Security 23)*. Anaheim, CA: USENIX Association, Aug. 2023, pp. 141–158. [Online]. Available: https://www.usenix.org/conference/usenixsecurity23/presentation/yang-zhuolin

[54] Y. Xu, J. Heinly, A. M. White, F. Monrose, and J.-M. Frahm, "Seeing double: reconstructing obscured typed input from repeated compromising reflections," in *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security*, ser. CCS '13. New York, NY, USA: Association for Computing Machinery, 2013, p. 1063–1074.

[55] L. Zhuang, F. Zhou, and J. D. Tygar, "Keyboard acoustic emanations revisited," *ACM Trans. Inf. Syst. Secur.*, vol. 13, no. 1, Nov. 2009.

[56] A. Compagno, M. Conti, D. Lain, and G. Tsudik, "Don't skype & type! acoustic eavesdropping in voice-over-ip," in *Proceedings of the 2017 ACM on Asia Conference on Computer and Communications Security*, ser. ASIA CCS '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 703–715.

[57] J. Liu, Y. Wang, G. Kar, Y. Chen, J. Yang, and M. Gruteser, "Snooping keystrokes with mm-level audio ranging on a single phone," in *Proceedings of the 21st Annual International Conference on Mobile Computing and Networking*, ser. MobiCom '15. New York, NY, USA: Association for Computing Machinery, 2015, p. 142–154.

[58] J.-X. Bai, B. Liu, and L. Song, "I know your keyboard input: A robust keystroke eavesdropper based-on acoustic signals," in *Proceedings of the 29th ACM International Conference on Multimedia*, ser. MM '21. New York, NY, USA: Association for Computing Machinery, 2021, p. 1239–1247. [Online]. Available: https://doi.org/10.1145/3474085.3475539

[59] L. Cai and H. Chen, "Touchlogger: inferring keystrokes on touch screen from smartphone motion," in *Proceedings of the 6th USENIX Conference on Hot Topics in Security*, ser. HotSec'11. USA: USENIX Association, 2011, p. 9.

[60] P. Marquardt, A. Verma, H. Carter, and P. Traynor, "(sp)iphone: decoding vibrations from nearby keyboards using mobile phone accelerometers," in *Proceedings of the 18th ACM Conference on Computer and Communications Security*, ser. CCS '11. New York, NY, USA: Association for Computing Machinery, 2011, p. 551–562.

[61] H. Wang, T. T.-T. Lai, and R. Roy Choudhury, "Mole: Motion leaks through smartwatch sensors," in *Proceedings of the 21st Annual International Conference on Mobile Computing and Networking*, ser. MobiCom '15. New York, NY, USA: Association for Computing Machinery, 2015, p. 155–166. [Online]. Available: https://doi.org/10.1145/2789168.2790121

[62] Y. Liu and Z. Li, "aLeak: Privacy leakage through context - free wearable side-channel," in *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications*, 2018, pp. 1232–1240.

[63] Q. He, E. Yang, and S. Fang, "A survey on human profile information inference via wireless signals," *IEEE Communications Surveys & Tutorials*, vol. 26, no. 4, pp. 2577–2610, 2024.

[64] K. Ali, A. X. Liu, W. Wang, and M. Shahzad, "Keystroke recognition using wifi signals," in *Proceedings of the 21st Annual International Conference on Mobile Computing and Networking*, ser. MobiCom '15. New York, NY, USA: Association for Computing Machinery, 2015, p. 90–102. [Online]. Available: https://doi.org/10.1145/2789168.2790109

[65] M. Li, Y. Meng, J. Liu, H. Zhu, X. Liang, Y. Liu, and N. Ruan, "When csi meets public wifi: Inferring your mobile phone password via wifi signals," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '16. New York, NY, USA: Association for Computing Machinery, 2016, p. 1068–1079.

[66] S. Fang, I. Markwood, Y. Liu, S. Zhao, Z. Lu, and H. Zhu, "No training hurdles: Fast training-agnostic attacks to infer your typing," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 1747–1760. [Online]. Available: https://doi.org/10.1145/3243734.3243755

[67] E. Yang, S. Fang, I. Markwood, Y. Liu, S. Zhao, Z. Lu, and H. Zhu, "Wireless training-free keystroke inference attack and defense," *IEEE/ACM Transactions on Networking*, vol. 30, no. 4, pp. 1733–1748, 2022.

[68] E. Yang, Q. He, and S. Fang, "WINK: Wireless inference of numerical keystrokes via zero-training spatiotemporal analysis," in *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, ser. CCS '22. New York, NY, USA: Association for Computing Machinery, 2022, p. 3033–3047.

[69] 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)*. Anaheim, CA: USENIX Association, Aug. 2023, pp. 859–876. [Online]. Available: https://www.usenix.org/conference/usenixsecurity23/presentation/gopal

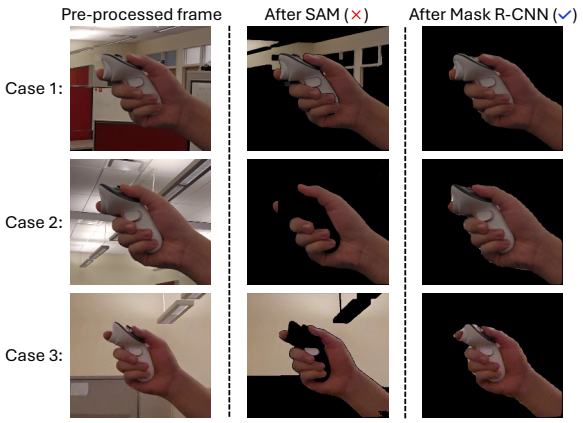


Fig. 31: Segmentation results of SAM and Mask R-CNN.

APPENDIX

A. Refined Hand Regions Extraction Comparison

Fig. 31 compares the segmentation performance between SAM and the trained Mask R-CNN via three exemplary cases, where SAM all fails and Mask R-CNN consistently succeeds. In general, SAM may introduce segmentation errors, including (1) *Case 1 - Redundancy:* the irrelevant background is recognized as part of the refined target hand area; (2) *Case 2 - Missing Important Details:* the irrelevant background is successfully winnowed out, but the controller is also removed;

Algorithm 1 Spatial Vector Generation

procedure S_VECTOR_GEN($[x_1, x_2, \cdots, x_N]$)
  for i in $\{1, 2, \cdots, N\}$ **do**
    $j \leftarrow$ ASSIGNCLUSTER(x_i) # Assign x_i to a cluster # based on similarity
    add x_i to C_j # $j \in \{1, 2, \cdots, M\}$
  end for
  for each C_j in Clusters **do**
    $r_j \leftarrow$ RANELEMENT(C_j) # Pick a random element # in each cluster
    add (C_j, r_j) to ElementList
  end for
  sort ElementList by r_j value
  SortedClusters $\leftarrow$ ElementList(:,1)
  for i in $\{1, 2, \cdots, N\}$ **do**
    for j in $\{1, 2, \cdots, M\}$ **do**
      if $x_i \in$ SortedClusters[j] **then**
        $y_i \leftarrow j$ # Assign the cluster index # after sorting
        break
      end if
    end for
  end for
  return $[y_1, y_2, \cdots, y_n]$
end procedure

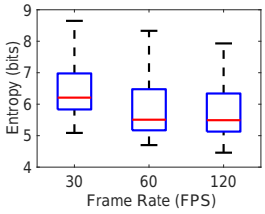


Fig. 32: Entropy vs. recording frame rate.

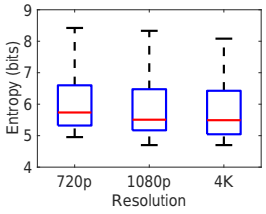


Fig. 33: Entropy vs. recording resolution.

(3) *Case 3 - Combination:* a big portion of the controller is discarded, while partial background are mistakenly segmented.

B. Algorithm for Deriving Spatial Vector

We define the spatial vector in Section IV-C1. Algorithm 1 outlines the spatial vector generation procedure, where the predefined function AssignCluster(x_i) classifies N elements into M clusters based on similarity, and RanElement(C_j) selects a random element from the cluster C_j.

C. Impact of Recording Frame Rates

Different frame rates of the recording can cause variations in the number of frames spanned across a keystroke event, which may affect the keystroke recognition accuracy. We compare the performance of three different common frame rates for the recording device - Samsung S23, including 30 FPS, 60 FPS, and 120 FPS. Fig. 32 shows the obtained entropy distribution under different frame rates. We see that regardless of the recording frame rate, *MotionDecipher* consistently reduces the entropies of various PINs to varying degrees. Moreover, the mean entropy slightly decreases as the frame rate increases. Specifically, the mean entropies at 30, 60, and 120 FPS are 6.53, 5.99, and 5.82 bits, respectively. This trend suggests that higher frame rates improve recognition accuracy by capturing more frames per keypress, thereby reducing entropy. *MotionDecipher* infers 70% of PINs at 30 FPS, 76% at 60 FPS, and 84% at 120 FPS within 100 guesses, further showing that higher frame rates enhance attack performance. Table IV shows the obtained average top-k accuracy. Notably, we see that the top-50 accuracy values for all frame rates exceed 0.6, indicating that with *MotionDecipher*, a large portion of PINs can be inferred within 50 guesses. Also, as the frame rate increases, the PIN entropy (i.e., strength) decreases, and the top-k accuracy rises accordingly, regardless of k.

TABLE IV: Average top-k accuracy vs. recording frame rate.

Frame Rate	Top-10	Top-25	Top-50	Top-100	Top-200
30 FPS	0.13	0.33	0.63	0.85	0.93
60 FPS	0.20	0.50	0.78	0.88	0.94
120 FPS	0.21	0.53	0.80	0.92	0.98

TABLE V: Average top-k accuracy vs. recording resolution.

Resolution	Top-10	Top-25	Top-50	Top-100	Top-200
720p	0.17	0.44	0.73	0.87	0.94
1080p	0.20	0.50	0.78	0.88	0.94
4K	0.21	0.52	0.78	0.89	0.95

D. Impact of Recording Resolutions

We consider three typical recording resolutions for the recording device - Samsung S23: 720p (1280×720 pixels), 1080p (1920×1080 pixels), and 4K (3840×2160 pixels). Fig. 33 presents the entropy distribution under varying recording resolutions, with all other factors held constant. We see our attack significantly reduces PIN entropy across all resolutions, with higher resolutions yielding slightly lower mean entropy. Particularly, the mean entropies for 720p, 1080p, and 4K are 6.18, 5.99, and 5.92 bits, respectively. Also, based on the inferred candidates for each selected PIN, we see that *MotionDecipher* infers 80% of PINs at 720p, 82% at 1080p, and 84% at 4K within 140 guesses. Higher resolutions often provide clearer recordings, enabling more accurate and richer spatiotemporal measurements, which in turn enhance overall inference performance. Table V shows the mean top-k accuracy remains comparable across all recording resolutions regardless of k, with slight improvements or stability as resolution increases. Notably, the top-50 accuracy is consistently at or above 0.73 across all resolutions.