Journal of Big Data

RESEARCH Open Access

Spoofing keystroke dynamics authentication through synthetic typing pattern extracted from screen-recorded video

Chrisando Ryan Pardomuan Siahaan* and Andry Chowanda*

*Correspondence:
chrisando.pardomuan@binus.edu;
achowanda@binus.edu

Computer Science Department, School of Computer Science, Bina Nusantara University, 11480 Jakarta, Indonesia

Abstract

As the inter-connectivity in cyberspace continues to increase exponentially, privacy and security have become two of the most concerning issues that need to be tackled in today's state of technology. Therefore, password-based authentication should no longer be used without an additional layer of authentication, namely two-factor authentication (2FA). One of the most promising 2FA approaches is Keystroke Dynamics, which relies on the unique typing behaviour of the users. Since its discovery, Keystroke Dynamics adoption has been continuously growing to many use cases: generally to obtain access to a platform through typing behaviour similarity, into a continuous keystroke monitoring on e-learning or e-exams platforms to detect illegitimate participants or cheaters. As the adoption of Keystroke Dynamics continues to grow, so does the threats that are lurking in. This paper proposes a novel exploitation method that utilizes computer vision to extract and learn a user's typing pattern just from a screen-recorded video that captures their typing process. By using a screen-recorded video, an attacker could eliminate the needs to inject a keylogger into the victim's computer, thus rendering the attack easier to perform and more difficult to detect. Furthermore, the extracted typing pattern can be used to spoof a Keystroke Dynamics authentication mechanism with an evasion rate as high as 64%, a considerably alarming rate given the impact it yields if the attacks are successful, which allows an attacker to pretend, mimic, and falsely authenticate as the victim (i.e., total account takeover). This paper also shows that from a screen-recorded video, one can produce a staggering statistical similarity in keystroke timing patterns as if they used an actual keylogger, and the extracted patterns can be potentially used to spoof the Keystroke Dynamics authentication service. To the author's best knowledge, there is no precedence of previous research that suggests this kind of attack (i.e. using video to spoof keystroke dynamics). This research can be used as the baseline for future research in this area.

Keywords: Behavioral-based exploitation, Computer vision, Cybersecurity, Keystroke dynamics

Introduction

Password-based authentication, the most widely-applied authentication mechanism, relies on something that supposedly only the legitimate user knows about, thus going by the term *knowledge factor*. However, many publications [1–3] have shown that some

© The Author(s) 2022. **Open Access** This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit http://creativecommons.org/licenses/by/4.0/.

attacks can be conducted to obtain or steal the password of a user; hence making it less reliable as an authentication method. Therefore, a concept of adding extra layers of verification into the authentication process, called two-factor authentication, was proposed and becoming widely adopted in many computer applications. One of the two-factor authentication mechanisms that have been extensively researched is called Keystroke Dynamics, which consists of a verification of password (*knowledge factor*) combined with the typing behaviour of the user when they type the password (*biometric factor*). Keystroke Dynamics mechanism has been implemented in several commercial services around the world, and some companies such as KeyTrac[1] and TypingDNA[2] have also provided Keystroke Dynamics authentication as Authentication-as-a-Service (AaaS).

However, the Keystroke Dynamics authentication mechanism can be exploited if an attacker manages to record and learn the typing pattern of the target user. Silently capturing and recording a user's typing pattern can usually be done using *keylogger*, a software that records the keys struck on a keyboard, typically covertly, so that a person using the keyboard is unaware that their actions are being monitored. Nevertheless, installing a *keylogger* into the victim's computer is challenging because it requires the attacker to construct a malware infection, manipulate the victim into running a malicious program, or even obtain physical access to the victim's computer in order to install the program manually. Throughout the decades, more covert Keystroke Dynamics exploitations have also been proposed, such as [4–6] where the authors identify certain users whose typing patterns are naturally similar to the others, then use their typing pattern to generate a synthetic pattern that matches other users' pattern. Other experiments [7, 8] discovered that an adversary typing information leaked from video frames could be fed to various machine learning models (e.g., Random Forest and Neural Network), trained on public datasets of inter-keystroke timing sequences, to infer the passwords or PINs typed by the victim. Nevertheless, many of those proposed methods require deliberately massive datasets of typing patterns to be collected by the attacker, making it less reliable and more difficult to do. To the best author's knowledge, only a few methods proposed that could entirely eliminate the need for huge keystroke datasets and active interaction with the victim altogether.

This paper proposes a novel method to extract a user's typing pattern using only a screen-recorded video that displays the user's typing process in an application. We discovered that by following and detecting a text-cursor object that indicates the newly-typed character out of the video frame, the delay between every typed character could be synthetically measured, so as the typing pattern of the user. Using a screen-recorded video, an attacker could eliminate the need to actively interact with the victim's computer or the victim itself, thus rendering the attack easier to perform and more difficult to detect. Furthermore, the extracted typing pattern can be used to spoof a Keystroke Dynamics authentication mechanism, which allows an attacker to pretend, mimic and falsely authenticate as the victim. In summary, the contributions in this paper are as follows: an algorithm is proposed to track and identify text cursors from video frames using computer vision, a novel exploitation method is designed to extract and generate a

[1] https://www.keytrac.net/.

[2] https://www.typingdna.com/.

synthetic typing pattern of the user a from screen-recorded video, and an attack simulation and statistical comparison is conducted between the extracted typing pattern and the user's actual typing pattern to conclude the susceptibility of Keystroke Dynamics authentication to the proposed attack. To the author's best knowledge, there is no precedence of previous research that suggests this kind of attack (i.e. using video to spoof keystroke dynamics). This research can be used as the baseline for future research in this area.

Recent work

Keystroke dynamics

Keystroke Dynamics is often defined as a process of observing and measuring a user's typing pattern on digital devices through a computer keyboard or touch screen panel [9]. The observed typing pattern is believed to be rich in cognitive qualities [10] and very unique so that it is possible to be used to distinguish a user from the others [11]. Therefore, Keystroke Dynamics is commonly utilized as an extra layer of the authentication process, namely biometric or behaviour-based authentication. In terms of observing and capturing the user's typing pattern, there are two types of Keystroke Dynamics authentication; *continuous observation* and *phrase-based observation*.

In continuous observation, a user's keystrokes are continuously monitored, and the signature is constantly analyzed [12]. The aim of continuous observation is usually to verify the identity of a user during the entire session on a computer [9]. Particularly, continuous observation is interested in knowing when an impostor is detected as long as user interaction with the system through input devices persists. Continuous observation is also known as dynamic authentication, and the implementation can be applied for account activity monitoring [13] and online examination [14]. In phrase-based observation, keystroke monitoring only occurs when a user is asked to type certain words in a password-based login scenario. Phrase-based observation is also known as static authentication, that means verifying a user at the very beginning of user interaction with the system [9] as soon as possible. The implementation of phrase-based observation includes physical access control [15], Automatic Teller Machine [16], and password sharing prevention [17]. More recently, phrase-based observation of keystroke dynamics has also been utilized for a more broad range of objectives other than for authentication, such as reinforcing captcha verification to prevent phishing attacks [18], detection of insider threats by recognizing the threat actor's stress level [19], or a mere wide-spectrum emotion detection [20], both through analyzing the user's typing behaviours.

The most commonly-used features in Keystroke Dynamics are as follows:

1 Hold Latency (HL)
   The elapsed time between the press and release of a keystroke. Hold Latency feature is also known as Dwell Time [9].
2 Inter-key Latency (IL)
   The elapsed time between the release of a keystroke and the press of the next keystroke. Inter-key Latency is also known as Down-Up Time (DUT) [21].
3 Press Latency

The elapsed time between two consecutive keystrokes' press events. Press Latency feature is also known as Down-Down Time (DDT) [21].

4 Release Latency

The elapsed time between two consecutive keystrokes' release events.

Various studies have proposed several approaches to authenticate users based on their typing patterns. The most widely-used approaches include Manhattan Distance [22, 23], Euclidean Distance [24], and k-Means [25]. Meanwhile, the effectiveness of Keystroke Dynamics authentication is measured based on the ability of the system to distinguish between real users and other users (or perpetrators) based on their typing patterns. Several evaluation metrics are commonly used, including False Acceptance Rate (FAR), False Rejection Rate (FRR), and Equal Error Rate (EER). According to [26], Authentication-as-a-Service (AaaS) applications such as KeyTrac, Behaviosec and TypingDNA would typically aim for the equilibrium between the FAR and FRR value. Meanwhile, applications that need to ensure absolute security would try to achieve higher FRR and lower FAR, and applications designed for forensic investigations might aim for lower FRR but higher FAR. An experiment shows that from a data set obtained from ten participants, the KeyTrac authentication system obtains an EER of 0.10, while Behaviosec obtains a slightly lower EER of 0.20 [27]. The lower the EER value, the more accurate the Keystroke Dynamics authentication system. As a basis of applicability, the utilization of typing biometrics and keystroke dynamics has been approved by New York State DMV [28], and European Banking Authority [29] as one of the compliant methods of identity authentication.

Keystroke exploitation

Two scenarios are generally used in various attack methods against keystrokes and Keystroke Dynamics: zero-effort attacks and non-zero-effort attacks. Zero-effort is a type of attack where one user's typing pattern data is matched with another user's typing pattern data so that if the two users have similar typing patterns, keystroke dynamics will treat the two users as the same user. It is called zero effort because the attack does not require much effort in impersonating the victim. Meanwhile, a non-zero-effort attack occurs when an attacker attempts to gain access to a system by actively impersonating the victim, such as through keylogger [4, 21], video-based pattern extraction that can be used for keystroke inference mechanism [8] or as a typing behaviour mimicry attack [30].

In [21], a tiny USB module that can be embedded in a user's keyboard is proposed, thus making the user's typing pattern recordable without being noticed. These typing patterns can be used to insert malicious keystrokes into the victim's computer autonomously based on the user's actual typing pattern, thus allowing attackers to evade Keystroke Dynamics security detection that may exist on the computer. The proposed method obtains evasion rates of +-90% on KeyTrac, +-85% on TypingDNA, and 100% on DuckHunt. Thus, it can be concluded that keystroke dynamics authentication can be tricked with sufficient typing pattern information. However, the utilization of this attack requires the perpetrator to install a malicious keylogger hardware device inside the victim's keyboard. Although the keylogger device was designed to be very small and compact, this attack still requires an active interaction with the victim's computers, thus becoming almost impossible to be performed remotely.

In [7, 8], they noticed that a user who logs into a system using a PIN or password receives direct feedback about the button pressed in the form of a dot (·) or asterisk (*) symbol and they recorded those symbol appearances on a video with a frame rate of 120fps. The recording was then processed with OpenCV to extract the symbols from each frame. When the symbols are detected, the corresponding frame number is recorded. Furthermore, to predict the password or PIN that is typed based on the data that has been collected previously, this study uses two algorithms, the Random Forest algorithm and Neural Network, using the inter-keystroke timing (DDT delays) that is inferred by the difference of corresponding frame numbers where each symbol appeared. As a result, the proposed method reduces the number of attempts to predict the correct password by 25% from a password list, with some passwords being guessed in just 19 attempts.

However, as the authors of [7, 8] relied on public keystroke datasets to train the models, the need for a massive typing datasets is then unavoidable. If the perpetrator cannot obtain a huge number of keystroke records specific to their victim, they have no other options but to use the publicly-available datasets. In our opinion, an attack would be more devastating as it evolves towards the least complexity, hence eliminating the needs of the reasonably huge datasets must be necessary. In addition, the main difference between [8] method and our proposed method gravitates around the type of symbols that can be detected in order to extract the typing pattern. Since [7, 8] focused more on extracting patterns of password and PIN typing, the symbols that can be detected are limited to dots (·) and asterisks (*) symbol. Meanwhile, our proposed method is able to extract a wider range of typographical symbols (i.e., lowercase A to Z and SPACE characters), hence the typing characteristics between every adjacent (bigram) characters can be recognized and extracted from the video.

Nevertheless, the objective of keystroke exploitation methods is mostly to manipulate a Keystroke Dynamics authentication into thinking that the attacker is a legitimate user, hence evaluated by *evasion rate*, or how many attempts of attack successfully identified as a legitimate user by the Keystroke Dynamics authentication; and usually indicated by the increasing value of False Acceptance Rate (FAR) obtained by the authentication system. The other common objective of the exploitation is to infer a piece of sensitive information (such as a password) from the text typed by the victim, hence evaluated by the rate of *search-space* successfully reduced to find the right password. For a broader perspective, Table 1 introduces and summarized some of the existing keystroke dynamics exploitation methods. To the author's knowledge, there has not been any attempt to extract a user's typing pattern solely from a screen-recorded video to spoof keystroke dynamics authentication.

Proposed architectures

To be able to extract typing patterns from a screen-recorded video, we need to be able to know precisely the timing when a character is typed and when the next character is typed afterwards; this is similar to a *keylogger* program. Furthermore, two challenges need to be solved. First, we need to follow the occurrence of the text cursor; a vertical line (|) appears on the screen that indicates where new text starts when the user begins to type in the application. We propose the text cursor to be followed because as

Table 1 More examples of existing Keystroke Dynamics attacks

Publications	Medium	Objective	Evaluation method	Evaluation result
Malboard [21]	Malicious USB keylogger installed on the victim's computer keyboard	Steal a victim's typing behavior via malicious USB keylogger and use them to impersonate the victim on a keystroke dynamics authentication	Evasion Rate (ER)	KeyTrac: ± 90% ER TypingDNA: ± 85% ER DuckHunt: ± 100% ER
SILK-TV [8]	A video that records physical screen (i.e., ATM and Computer screen) which displays password/pin input	Extract passwords and PINs typing delays and use them to infer the plaintext behind the masked password and PINs	Reduced Search-space	Reduced the password's search space by 25% to 385% depending on the complexity of the password
Mimicry [30]	A video that records the victim's typing activities and their finger movements on a smartphone	Extract a victim's typing behaviour by observing the fingers' movements and create an interface for an attacker to mimic the typing behaviour	Evasion Rate (ER)	± 97% ER for ≤ 3 attack attempts against Touchalytics
EyeTell [31]	A video that records the victim's face and gaze while typing their PIN on a touch-screen device	Extract a victim's keystrokes by capturing and analyzing his eye movements and use them to infer the typed PINs	Reduced Search-space	4-digit PIN: 74% of the PINs are located in the Top-10 PIN wordlist 6-digit PIN: 80% of the PINs are located in the Top-10 PIN wordlist

it indicates where the next character will be typed, it can be used to capture and identify the last character that has been typed by the user from every frame in a video sequentially. Therefore, identifying both the occurrence and the position of a text cursor on the screen enables us to extract that newly-typed character out of the video frame and calculate the timing when that character appears. Second, after the text cursor can be detected and tracked, we need to correctly extract the last character typed out of the entire video frame and convert it into ASCII value to be processed digitally. That is the second challenge, and to accomplish that, we propose an algorithm to automatically generate a tight bounding box region around the last character typed on each frame, then use the ResNet algorithm to recognize the character isolated inside that bounding box to convert it to ASCII value. Figure 2 illustrates the flow of the proposed methodology in this research. In hindsight, our proposed approach follows a similar base method with [7, 8], but our objectives are different as we focused on spoofing the keystroke dynamics, while their methods are more on how to extract the user's typing from the leaked inter-keystroke timings.

Text cursor detection and tracking

As mentioned before, the first proposed approach is designed to detect and track the text cursor appearance on a screen-recorded video, as can be seen on Algorithm 1. The proposed algorithm is built using OpenCV [32].

Algorithm 1 Text Cursor Detection Algorithm

$prevFrame \leftarrow readFrame()$
$prevContour \leftarrow None$
while $True$ **do**
  $frame \leftarrow readFrame()$
  $frame \leftarrow cannyEdgeDetection(frame)$
  $frame \leftarrow bitwiseXor(frame, prevFrame)$
  $contour \leftarrow getContour(frame, prevFrame)$
  $x1, y1, w1, h1 \leftarrow boundingRect(contour)$
  $lineTip \leftarrow getLineTip(contour, prevContour)$
  if $lineTip \neq None$ **then**
    $prevContour \leftarrow contour$
    $prevLineTip \leftarrow lineTip$
  else
    $lineTip \leftarrow prevLineTip$
  end if
  $xmin, ymin, xmax, ymax \leftarrow getRect(lineTip)$
end while

Initially, frame f is read from the screen-recorded video. Then, the Canny algorithm detects edges and extracts structural information of every object (characters, buttons, text cursor, and others) appearing on the frame. After that, a bitwise XOR operation [32] is applied between the extracted frame f and the previous frame $f - 1$ on the video. With this approach, we can extract every object (the moving text cursor hypothetically) that changed position since the previous frame $f - 1$. Next, the moving objects (including the text cursor) are extracted by getting their contours. Finally, the text cursor object is identified and extracted by comparing the shape of the contour with the pre-defined average shape of a text cursor. Then, a bounding box region is calculated and generated around the object, producing the text cursor object's size and position. The text cursor

Fig. 1 An illustration of text and text cursor on Microsoft Word

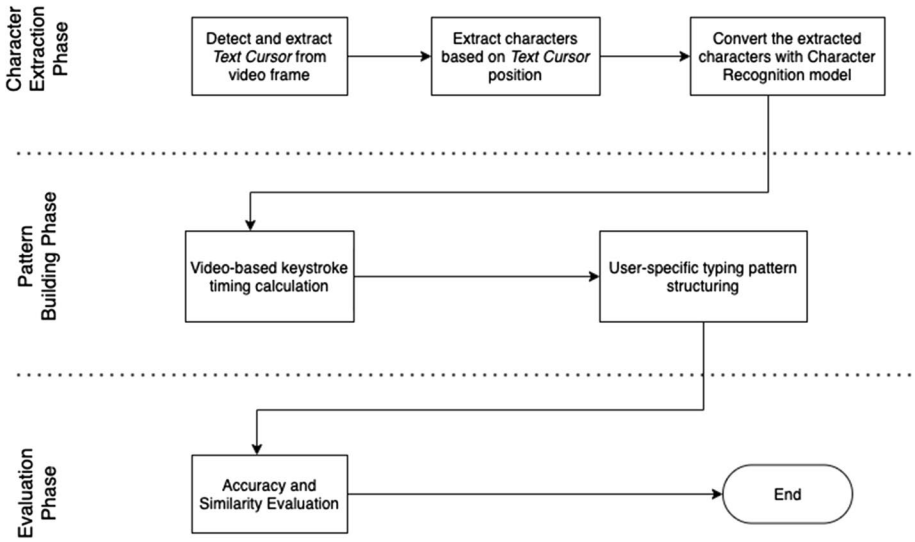

Fig. 2 Flowchart of the proposed method

position is represented with *xMin*, *yMin*, *xMax*, and *yMax* coordinates. The bounding box that surrounds the text cursor object is called Cursor Bounding Box (CBB) in this research.

Character isolation and timing extraction

After the coordinates and the CBB region of the text cursor are obtained, an algorithm is run to extract the most recently typed character from each frame based on the obtained CBB coordinates. This approach relies on the assumption that the most recently typed character by the user must be the character to the left of the text cursor object (Fig. 1). For simplicity, we called the most recently typed character in the frame as Rightmost Character (RC). Figure 2 shows a flowchart that illustrates the proposed approach for character extraction.

First, we use CBB coordinates to estimate the font size of the characters recorded on the screen. To do that, initially we calculate the height of the CBB (Eq. 1), then use the height to estimate the font size (Eq. 2). This can be done because the height of the text cursor will grow accordingly as the font size on the screen gets bigger.

$$Cursor\ Height = CBB_{ymax} - CBB_{ymin} \tag{1}$$

$$Font\ Span = \frac{Cursor\ Height}{PPI} \times 72 \tag{2}$$

In the above equations, PPI stands for pixels per inch, which is the screen specification where the screen-recorded video is captured. The constant value 72 is the ratio of the font size to screen size (in inches) according to the following Eq. 3 [33].

Fig. 3 IBB and CBB Illustration on Microsoft Word

$$1pt = \frac{1}{72} \times inch \tag{3}$$

To perform character isolation based on the previously-captured text cursor object, first the system calculates the x-axis midpoint of the CBB region (Eqs. 4 and 5).

$$Cursor\ Width = CBB_{xmax} - CBB_{xmin} \tag{4}$$

$$CBB_{xcenter} = \frac{Cursor\ Width}{2} \tag{5}$$

After that, the system generates a second bounding box region, namely the Isolation Bounding Box (IBB), that surrounds an estimated two or three characters to the left of the text cursor object and then extracts the Rightmost Character out of those captured characters. This is needed to accommodate the probability that characters might differ in width, so creating a bounding box that surrounds the character area on the exact basis might be impossible. The IBB coordinates, in the form of *xMin*, *yMin*, *xMax*, and *yMax*, is calculated according to Eqs. 6 to 9.

$$IBB_{xmin} = CBB_{xmin} - (2 \times Cursor\ Width) \tag{6}$$

$$IBB_{ymin} = CBB_{ymin} \tag{7}$$

$$IBB_{xmax} = CBB_{xmax} + (2 \times Cursor\ Width) \tag{8}$$

$$IBB_{ymax} = CBB_{ymax} \tag{9}$$

Figure 3 illustrates the Isolation Bounding Box (IBB) drawn based on the isolation coordinates that have been generated. We can see that the resulting IBB is drawn right next to the text cursor displayed by the application (Microsoft Word).

The identification and extraction of the Rightmost Character from the IBB region are made using the Connected Component Analysis (CCA) algorithm. Figure 4 illustrates an Isolation Bounding Box (IBB) area and the characters that have been successfully separated from the bounding box. This research performs image segmentation with CCA 4-connectivity to reduce the time complexity required. For simplicity, every separated character from the bounding box is labeled as the Isolated Character (IC). Based on the observations made, and an IBB processed using CCA can have the following components:

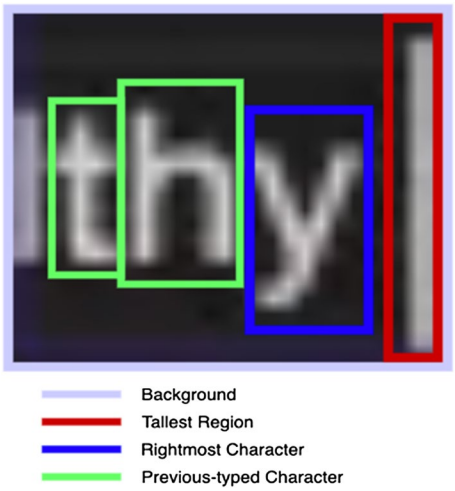

Fig. 4 Region segmentation in an isolation bounding box (IBB)

1 Background Region (BR)
BR is always the first order component identified by the CCA algorithm; it indicates the background area of the objects (characters) inside the IBB area.
2 Tallest Region (TR)
TR is a component that has the largest height in an IBB region and indicates a text cursor object because generally, text cursors must be the tallest object compared to other characters or letters.
3 Rightmost Character (RC)
RC is an Isolated Character component located in the rightmost position of the IBB but not the Tallest Region. This component contains the most recently typed character of the frame.
4 Previous-typed Character (PC)
PC is an Isolated Character component (or several ICs) located to the left of a Tallest Region component, is not a Rightmost Character, and does not intersect with the Isolation Bounding Box's vertical and horizontal borders. These components indicate the residue of characters that the user has typed in the previous frames.

In general, the Tallest Region component is never used and only was classified through CCA to identify other components such as the Rightmost Character. However, when the resolution of the screen-recorded video is non-optimal, the character located right to the left of the text cursor object might be merged with the text cursor object itself. This non-optimal condition, namely the Sticking Characters condition, is illustrated in Fig. 5. Therefore, the Tallest Region component must undergo a character separation process to separate the most recently typed character (Rightmost Character) from the text cursor object. To identify if a Tallest Region component contains Sticking Character, we rely on the unusually larger widths that a Tallest Region component might have (e.g., the text cursor object's width should only be 2px, but the Tallest Region width is 12px). Should that happen, it is assumed that the Tallest Region component contains a Rightmost Character that's stuck to the text cursor object.

Fig. 5 Sticking characters problem, where a character seems visibly merged with text cursor due to low-resolution frame

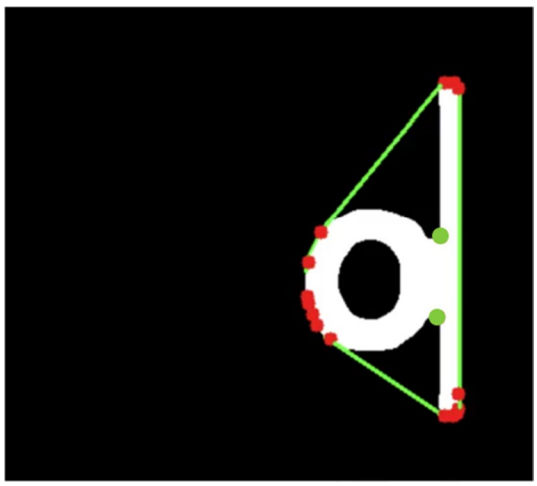

Fig. 6 Convexity defects for sticking characters Separation. Green Dot for separation point, Red Dot for non-separation point

The character separation process implements the Convexity Defects algorithm to determine points where the separation must be done. If a defect is detected with a hull-to-defect distance larger than a certain threshold, the defect coordinates are stored as potential separation points. In this research, the hull-to-defect threshold is set to 20 pixels, while the close defects (non-separation points; e.g., defects that happen because of rough pixels) usually have a hull-to-defect distance of around 1 to 3. Comparing all the potential separation points collected before, we select the coordinates that are located on the far right as we want the separation to be performed as close to the text cursor object as possible; under the assumption that the text cursor object must be located on the right side of the frame. Figure 6 illustrates an example of a character with separation (green circle) and non-separation (red circle) points respectively. The circle marker in the Figure shows the position of the points.

After that, the IBB area will be cropped up to 5 pixels (as a margin of error) before the coordinates of the selected separation point. The cropped area, which supposedly contains the clean Rightmost Character separated from the text cursor object, is returned for further processing, considering the separation is successful. Figure 7 shows an example of a crop-box generated according to the selected separation point. In hindsight, the area to the left of the separation point will contain the

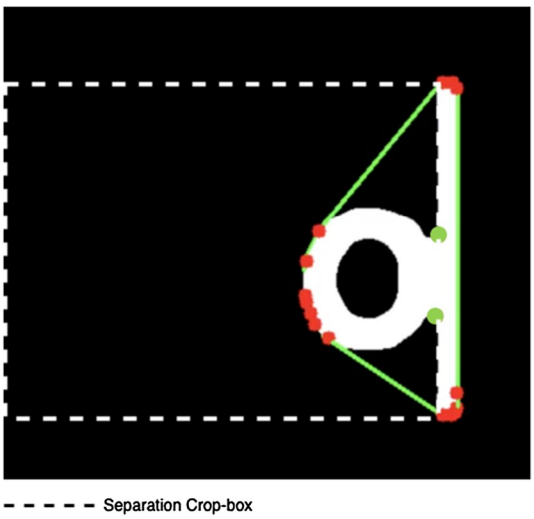

Fig. 7 Separation Crop-box for Sticking Characters Separation, areas inside the dotted line contains the character without the Text Cursor

clean Rightmost Character; the area to the right of the separation point will contain the text cursor object. Furthermore, Rightmost Character, Previous-typed Character, and even the Tallest Character components (the Isolated Characters) that have gone through the separation process are passed to a character recognition model that will be extensively explained in the next section.

Character conversion and timing extraction

ResNet50 architecture [34] is used in this research as a character recognition model as it uses 'shortcut connections' to solve the vanishing gradient problem. In this paper, a deeper network is necessary to accommodate different fonts styles and sizes. Nevertheless, as the network depth increases, there is a higher chance that the model could go over-fitting, indirectly affecting the accuracy of the extracted timing for the typing pattern. Therefore, we need a network that can optimally afford to have deeper layers, yet is not computationally expensive and still provides great accuracy. Hence, in this research, we picked ResNet50 architecture.

Training and evaluating recognition model

To train and evaluate the ResNet50 architecture, 4200 images representing A to Z characters (both uppercase and lowercase) are used. The images contain characters in two different font families: Calibri and Helvetica Neue (both are Proportional fonts). The dataset is split into Training and Testing datasets with an 80:20 ratio. The model is trained using Python with the Keras library (Table 2).

The training process is conducted in 40 epochs, the learning rate is set to 0.1, and the batch size is 64. As for the evaluation process, the loss will be calculated using the Categorical Cross-entropy function as the model aims for the multi-class classification task. Stochastic Gradient Descent (SGD) will be used as the optimizer to maintain less computational expensiveness as we aim to propose a system that works as close to real-time

Table 2 ResNet hyperparameters

Parameter	Value
Learning rate	0.1
Batch size	64
Epoch	40
Optimizer	Stochastic gradient descent
Loss	Categorical ross-entropy

as possible. For most of the captured character frames are relatively small in terms of resolution, we implemented the ResNet architecture to use 32x32 images; pixel intensities in the images are also scaled down from (0, 255) to (0, 1). The training and evaluation result will be extensively described in "Result and discussion" section.

Character conversion with recognition model

The Isolated Characters (ICs) obtained from the previous processes are sequentially fed into the ResNet-based character recognition model. The model will then attempt to recognize the character inside the IC frame and return the prediction result and the confidence value to be processed further. When the model fails to identify a character from the IC frame prominently, one of the two following fallback scenarios will be run. If the confidence obtained after the prediction is lower than 70%, the confidence threshold set in this research, then the detected character will be set to [*low_confidence*]. On the other hand, if the model detects no character on the IC frame, the detected character will be set to [*no_character*], and the result will not be processed further.

Keystroke timing extraction and structuring

From each IC frame that has been processed through the character recognition model, the recognition result will be added into a data structure. There are two sets of data used in this research, goes by the terminology:

1 KUnit
   A representation unit containing a single character has been successfully separated from a frame (IBB area) and has been converted to ASCII through the character recognition process.
2 KeystrokePoint
   A set of KUnit is detected from different frames (IBB area), representing the same characters that the user typed.

KeystrokePoint is used to store several KUnits that represent the same characters that the user types. This is important because the same character can appear over several different frames throughout the video. However, because it appears on different frames, the proposed approach will create different KUnits for that same character, even though we must process them simultaneously as a single character data. For example, character a first appears on the frame f. Then, not until frame $f + 5$ does the character b is typed by the user and appear on the frame. That means, character a will appear in at least 5

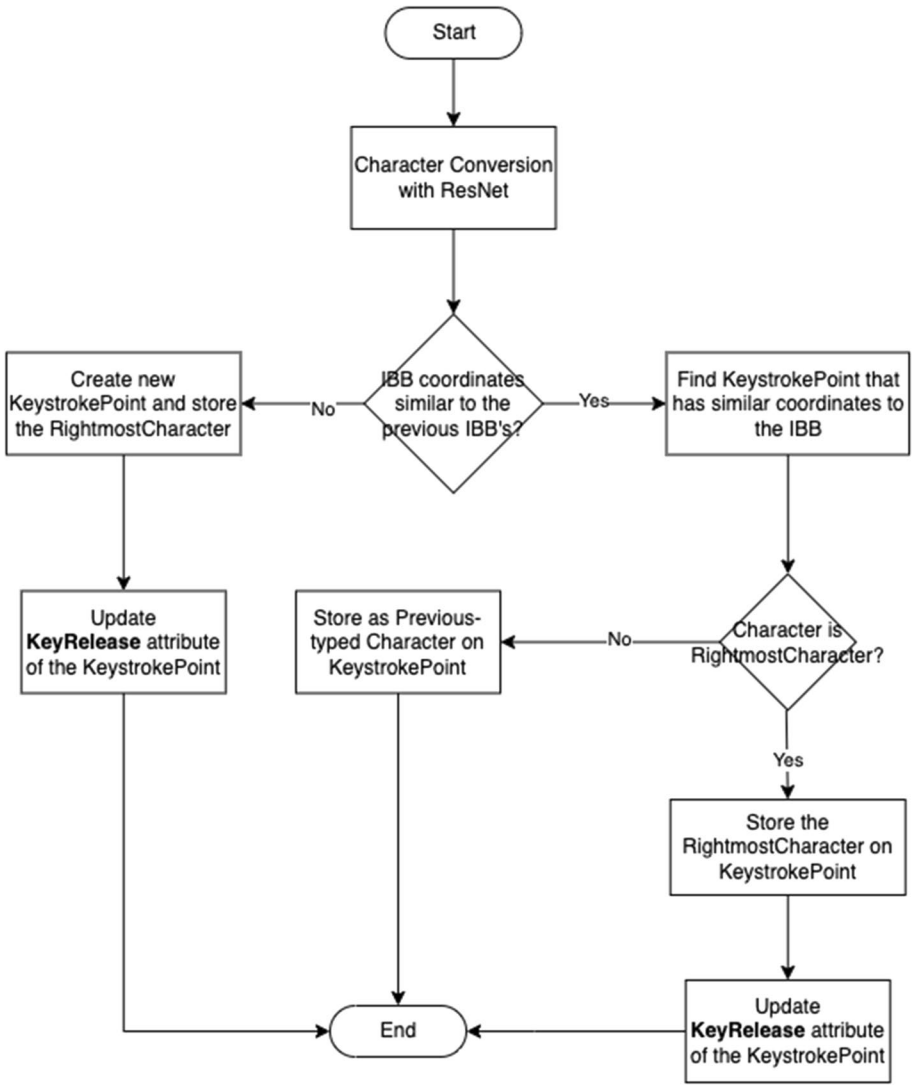

Fig. 8 Storing KUnit into KeystrokePoint

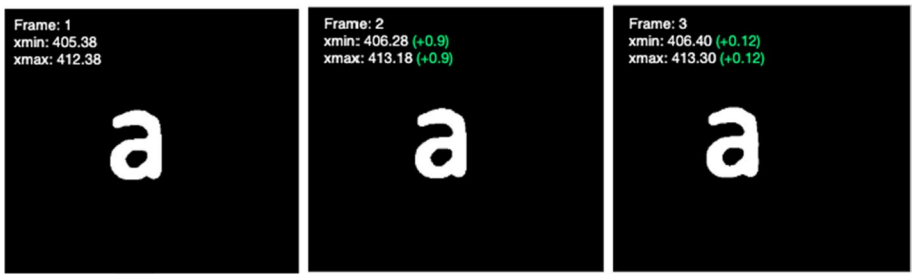

Fig. 9 KUnits on different frames with similar coordinates (same KeystrokePoint)

Table 3 KeystrokePoint attributes

Attribute	Data type	Description
KeyPress	Integer	Index of frame where the character first appears
KeyRelease	Integer	Index of frame where the character last appears
KeyDelay	Float	Represents Down–Down Time (DDT)
KeyText	Character	Identified character from the IC frame
OCR Confidence	Float	Confidence score of character recognition process

Table 4 KUnit attributes

Attribute	Data type	Description
FrameNo	Integer	Index of frame where the KUnit originates
KUnit Image	Binary	The binary data of KUnit image
Shape	Float	The shape of the KUnit (width, height)
X_Coord	Float	The x-axis position of the KUnit on the frame (xmin, xmax)
Y_Coord	Float	The y-axis position of the KUnit on the frame (ymin, ymax)

frames (frame f to frame $f + 4$). To do so, several KUnits which have similar coordinates (we use 5 pixels threshold in this research) will be considered as the detection result of the same one character (Fig. 9) and combined into one KeystrokePoint. Tables 3 and 4 illustrates the attributes contained in a KeystrokePoint and KUnit sequentially. The process of storing every KUnit into its appropriate KeystrokePoint is carried out as illustrated in Fig. 8.

Initially, after an IC frame is processed through the character extraction process and KUnit k is generated, the system will check whether a previously-recorded KeystrokePoint has coordinates similar to the KUnit k. If so, KUnit k will be added to KUnit collection inside that KeystrokePoint. Then, if KUnit k is the Rightmost Character or Tallest Region through the separation process, the *KeyPress* and *KeyRelease* attributes will be updated. Otherwise, the KUnit will still be added to the KUnit collection, but the KeystrokePoint's timing attributes (*KeyPress* and *KeyRelease*) will not be updated at all. On the other hand, if the system fails to find the previously-recorded KeystrokePoint with similar coordinates, a new KeystrokePoint will be generated.

Video-inferred Keystroke timing calculation

In estimating the user's typing pattern, the system calculates the Press Latency (Down-down Time), namely *KeyDelay*, from the available information based on the obtained and generated KUnits and KeystrokePoints. KeyDelay information will be generated based on the following formulations.

First, the system calculates the *InterKeystrokeTiming* (IKT) value, which indicates the number of frames in which a character appears as the Rightmost Character. *InterKeystrokeTiming* is calculated using the following formula:

$$Inter\ Keystroke\ Timing(IKT) = (Key\ Press - Key\ Release) + 1 \tag{10}$$

Table 5 Sample groups and predetermined sentence templates

ID	Type	Sentence
A	Password phrase	abudhabiacrossthesea
B	Greeting sentence	hi my name is [NAME]

where *KeyPress* and *KeyRelease* are the first and last frame numbers in which the character appears as the Rightmost Character. The result is then incremented by one because when a character appears and disappears in the same frame (*KeyPress* = *KeyRelease*), that character has been detected in 1 frame. Since the *InterKeystrokeTiming* calculation is based on the number of frames, the result needs to be converted to milliseconds (ms), the measurement unit commonly used in the keystroke dynamics authentication mechanism. The frame-to-millisecond conversion is done according to the following formula:

$$frame\ To\ Ms(fps, IKT) = \left(\frac{1000}{fps}\right) \times IKT \tag{11}$$

Assuming the screen-recording video runs at 30fps (30 frames per second), each frame should take about 33ms. For example, if a character appears as the Rightmost Character for four frames (IKT=4), the timing between the character is typed until the next character to be typed is approximately 133ms (4 frames × 33ms). After the IKT value is converted to milliseconds, that value can be used to represent the KeyDelay value:

$$Key\ Delay = frame\ To\ Ms(fps, IKT) \tag{12}$$

Result and discussion

Experimental settings

The experiment and evaluation process is done according to the following setup and requirements. First, the typing-record process is conducted on 14 different typing skills and fluency subjects who usually work with computers. Every subject performed the following typing tasks:

1. Password Phrase: every subject is given a predetermined password phrase and typed on a template document in Microsoft Word.
2. Greeting Sentence: every subject typed a predetermined greeting phrase followed by their name on a template document in Microsoft Word.

Table 5 shows the sample groups and the text template that needs to be typed by every subject. All characters must be typed in lowercase and as much as possible without typos. To prevent the emergence of environmental pressure, every subject may conduct training sessions to adjust themselves to the evaluation environment. Please note that by considering only the lowercase characters, we intentionally remove the special character typing combination from affecting the experiment result, since the action of pressing

Table 6 Experiment subjects demography

Subjects	Occupation	Age
S-001	Lab Technician	23
S-002 to S-004	Lecturer	27 to 40
S-005 to S-009	Software engineer	21 to 23
S-009 to S-014	Student	16 to 21

a special character (e.g., pressing SHIFT or CAPSLOCK to obtain an uppercase character) will not be visible on the screen, hence will be very difficult to extract. Though, the relevance of uppercase keystrokes (which usually consists of a combination between one character keystroke and one SHIFT/CAPSLOCK keystroke) as well as other special character keystrokes should also be studied in future experiments.

The screen-recording process is conducted using OBS software,[3] with video resolution of 720p (1280x270). The frame-per-second (FPS) value is set to a constant of 30fps. The pixels per inch (PPI) value is 220.53 PPI (MacBook Pro, macOS). A Python-based *keylogger* is set to capture the subject's actual typing pattern as the ground truth.

There are two types of samples collected in this experiment:

1 Emulated KeyDelay: is a delay between one key-press to another—extracted and inferred by the proposed method.
2 Actual KeyDelay: is a delay between one key-press to another – recorded with an actual keylogger. This is the ground truth sample.

To measure the quality of the extracted typing pattern, various statistical tests are used: the normality test is conducted with Shapiro and K-squared Test, Levene Variance Test is used to assess the equality of the variances, and Paired T-Test or Wilcoxson Test [35] to measure KeyDelay similarity on both samples. Meanwhile, to measure the performance and capability of the proposed method, the accuracy of the character recognition model is calculated, and the amount of time taken to process the video is also measured. The aim is to have an accurate character recognition result that works near-real-time (e.g., a 10-s video should take 10 s to be processed). Every aspect of the performance evaluation in this experiment, including the processing time measurements, was conducted on a MacBook Pro (2020) device with 1.4GHz Quad-Core 8th Gen and 8GB RAM 2133MHz. The evaluation is conducted separately on every sample group. The demography of the experiment subjects can be seen in Table 6.

Password phrase group

In this sample group, every subject typed a similar predetermined password string:

abudhabiacrossthesea

[3] https://obsproject.com/.

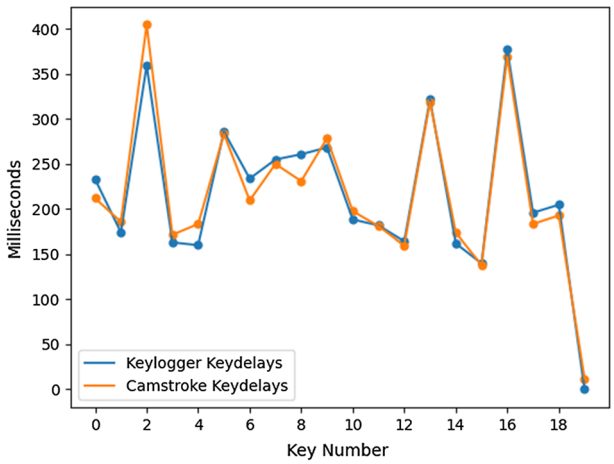

Fig. 10 KeyDelay comparison on password phrase sample group

The string is 20 characters long, alphabetical only, and contains no space. No typo occurred in the process of collecting every sample. This sample group represents a password input.

Similarity evaluation

Because every subject typed the same characters, we calculated the average of the Emulated KeyDelay and Actual KeyDelay from every sample. The average value for each character is plotted on Fig. 10 below. The plot shows that the proposed method could produce Emulated KeyDelay value similar to the Actual KeyDelay. Furthermore, statistical test on the samples also suggests that the data points are similarly close.

Normality test From 14 experiment subjects, the Emulated KeyDelay from 8 subjects (S-002 and S-003, S-007, S-009, S-010, and S-012 to S-014) are normally distributed (p-value > 0.05), while six others are not normally distributed. The same eight subjects also have a normal distribution for the Actual KeyDelay, while six other samples are not normally distributed. In addition, normality tests conducted on the averaged value of Emulated KeyDelay and Actual KeyDelay also suggest that both delay samples are normally distributed, with p-value = 0.3218 for Emulated KeyDelay and p-value = 0.3723 for Actual KeyDelay.

Variance test In this sample group, the samples from all experiment subjects have equality in variance (p-value > 0.05) between the Emulated KeyDelay and the Actual KeyDelay. In addition, the variance test conducted on the averaged Emulated KeyDelay, and Actual KeyDelay also suggests that both delay samples have equality in variance with p-value = 0.8040.

Mean similarity test From 8 samples (S-002 and S-003, S-007, S-009, S-010, and S-012 to S-014) that are normally distributed, Paired T-Test shows that there is every sample no significant mean difference between the Emulated KeyDelay and the Actual KeyDelay. Meanwhile, on six samples that are not normally distributed, Wilcoxon Test shows that there is also no significant mean difference between the Emulated KeyDelay and the Actual KeyDelay, except for subject S-011 that obtained p-value = 0.01575. Furthermore, Paired T-Test conducted on the averaged Emulated KeyDelay,

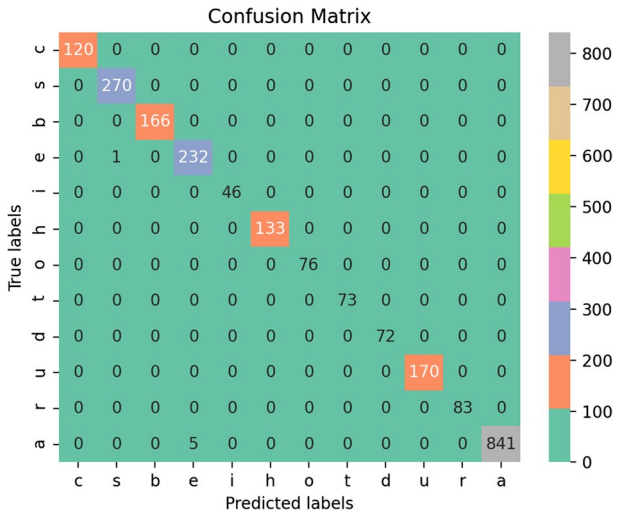

Fig. 11 Password phrase confusion matrix

and Actual KeyDelay (normally distributed) also suggests no significant mean difference between both samples.

Performance evaluation

In this sample group, 12 unique characters are typed by 14 subjects, in an overall of 2404 Isolation Bounding Box frames. The character recognition model can correctly recognize the characters with 94.64% accuracy. Figure 11 shows the confusion matrix of the character recognition model used in this experiment.

Furthermore, to obtain the performance of the proposed method, the time elapsed to process and extract the typing pattern from the screen-recorded video is measured. Fourteen screen-recorded videos are processed in this sample group with an average video length of 6.615 s. Therefore, the average processing time obtained from those 14 videos is 35.4385 s. That indicates the proposed method requires 5.35x processing time for every second on the video that needs to be processed.

Greeting sentence group

In this sample group, every subject typed a predetermined sentence followed by their name (without bracket):

hi my name is [NAME]

The text is alphabetical only and varies between 18 to 34 characters long (space-included) due to the subjects' different names. There is no typo that occurred in the process of collecting every sample. In addition, there are no two subjects that have a similar name. Thus the sentences typed by one subject with another subject are different. This sample group represents a dynamic input.

Similarity test

The captured delays are not averaged in this sample group since the experiment subjects typed a text that varies in characters. From 14 experiment subjects, the best performance

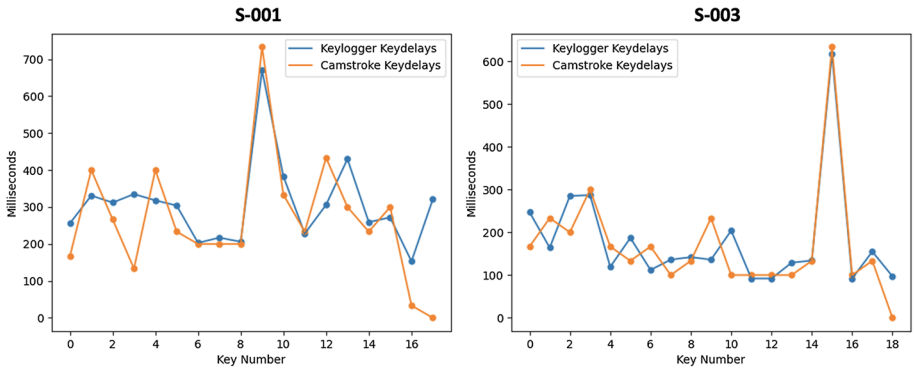

Fig. 12 Best KeyDelay comparison for greeting sentence sample group

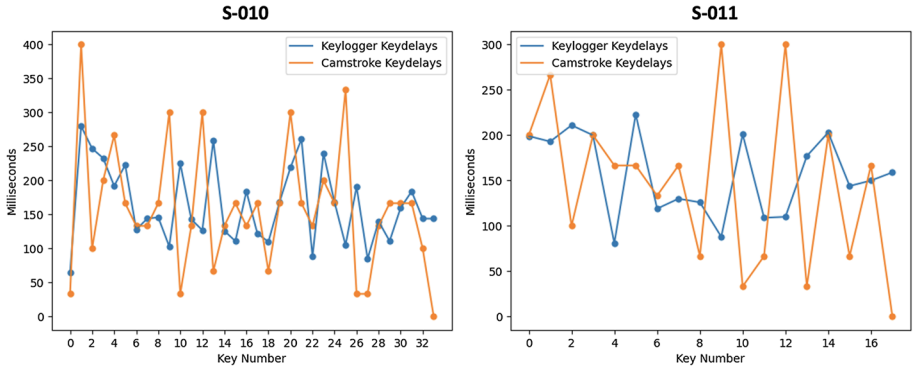

Fig. 13 Poor KeyDelay comparison for greeting sentence sample group

is obtained on samples from subjects S-001 and S-003. The comparison plot between the Emulated KeyDelay and the Actual KeyDelay for those subjects can be seen in Fig. 12.

Meanwhile, the worst performance is obtained on samples from subjects S-010 and S-011. The comparison plot between the Emulated KeyDelay and the Actual KeyDelay for those subjects can be seen in Fig. 13.

Although there are some significant differences in the delay value on specific points in the plot graph, further statistical tests conducted on the sample suggests that the proposed method can emit rather similar delays and capture the overall pattern of the subjects' typing, as explained below.

Normality test From 14 experiment subjects, the Emulated KeyDelay samples from 3 subjects (S-005, S-010, and S-011) are normally distributed (p-value > 0.05), while 11 others are not normally distributed. For the Actual KeyDelay samples, five subjects (S-001, S-005 to S-006, and S-010 to S-011) are normally distributed, while nine samples are not normally distributed.

Variance test The samples from all experiment subjects have equality in the variance (p-value > 0.05) between the Emulated KeyDelay and the Actual KeyDelay, except the sample from subject S-011 with p-value = 0.026261.

Mean similarity test From 3 samples (S-005, S-010, and S-011) that are normally distributed, Paired T-Test shows no significant mean difference between the Emulated KeyDelay and the Actual KeyDelay. Samples from subjects S-010 and S-011 obtained

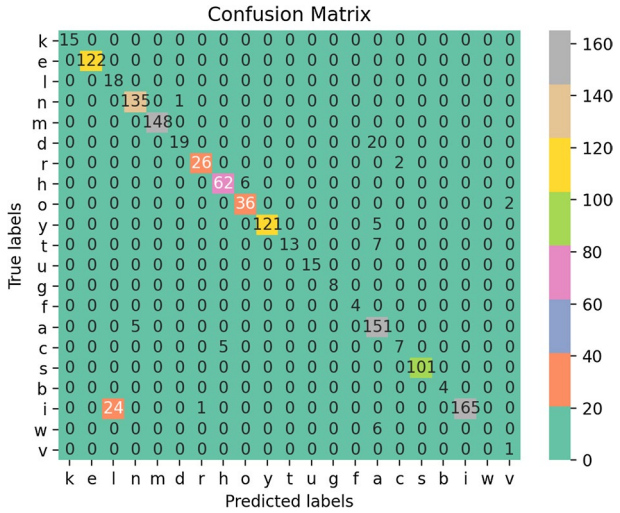

Fig. 14 Greeting sentence confusion matrix

Table 7 Text configuration for attack spoofing simulation

Mode	Evaluation group	Text
Password	Password phrase	abudhabiacrossthesea
Freetext	Greeting phrase	Hi my name is [NAME]

p-value 0.7191 and 0.6811 accordingly, although there are a few significant differences between the delays as shown in Fig. 13 previously.

On the other hand, on 11 samples that are not normally distributed, Wilcoxon Test shows that there is also no significant mean difference between the Emulated KeyDelay and the Actual KeyDelay, with p-value ranging from 0.1701 to 0.8076.

Performance test

In this sample group, 22 unique characters (space-included) are typed by 14 subjects, in an overall of 1599 IBB frames. The character recognition model can correctly recognize the characters with 90.78% accuracy. Figure 14 shows the confusion matrix of the character recognition model used in this experiment.

Moreover, the average video length of 14 screen-recorded videos was processed in 4.5 s. Therefore, the average processing time obtained from those 14 videos is 19.9729 s. That indicates the proposed method requires a 4.43x processing time for every second on the video that needs to be processed.

KeyTrac spoofing simulation

To inherently evaluate whether the proposed method increases the susceptibility of a Keystroke Dynamics authentication system against spoofing attacks, we designed a carefully-considered attack simulation against the KeyTrac authentication service. Off-the-shelf, KeyTrac supports two authentication modes: *password mode* and *freetext mode*.

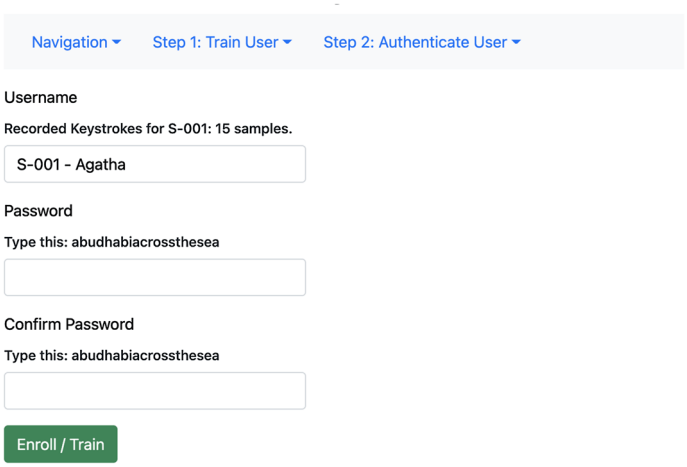

Fig. 15 KeyTrac recorder web application for attack simulation

Table 8 KeyTrac API configuration

Mode	Config	Value	Description
Password	Threshold	50%	Threshold for a user to be considered as authenticated
	Min. Sample Count	2	Minimum number of (valid) samples.
Freetext	Threshold	50%	Threshold for a user to be considered as authenticated
	Min. Text Length	10	Minimum length of a text to be used for an enrollment.

Texts that the participants are required to type are taken from the previous evaluation, as can be seen in Table 7.

In the beginning, the 14 subject participants are asked to register their typing pattern through a web application that utilizes KeyTrac API (Fig. 15) by typing a specific text. Then, they are asked to type the exact text in an attempt to authenticate themselves through KeyTrac. The API configuration applied to KeyTrac can be seen in Table 8.

KeyTrac will try to recognize the user based on their typing patterns and provide feedback on whether the user can be authenticated or not, along with the confidence score. This authentication step is carried out 15 times for each user. In total, a user will perform 30 text-typings across two authentication mods, with extra two typings for registration.

Biometrics evaluation measurement

The authentication result and confidence score of every user are taken to calculate the FRR value of the authentication system. The typing patterns of all subjects are also collected to be tested as a zero-effort attack to measure KeyTrac's baseline performance. Meanwhile, the Emulated KeyDelay of each user, obtained from the video analysis, are programmatically replayed on the web using a bot application to see whether KeyTrac also recognized them and authenticates the bot as a legitimate user. Zero-effort attacks using Emulated KeyDelay are also performed. In total, there are 196 samples (14 users $\times$ 14 zero-effort and authentic typings) used for baseline measurement, as well as 196 samples used for attack measurement (14 users $\times$ 1 spoofed typing + 13 zero-effort typings). Finally, the authentication results of the bot's typings are taken to calculate the

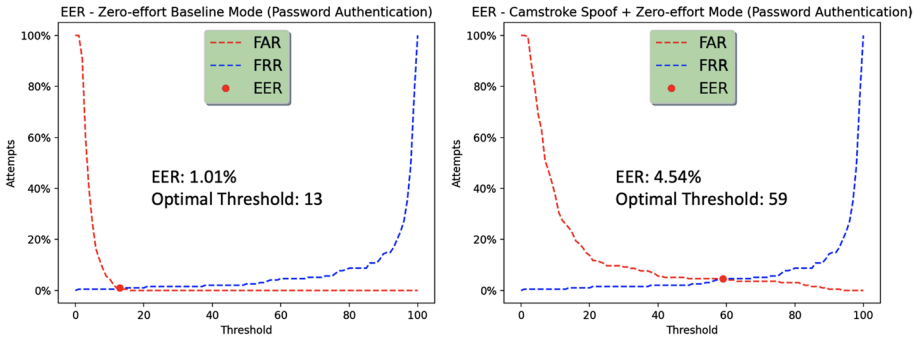

Fig. 16 EER comparison for password-mode, before attack (left) and after the attack (right)

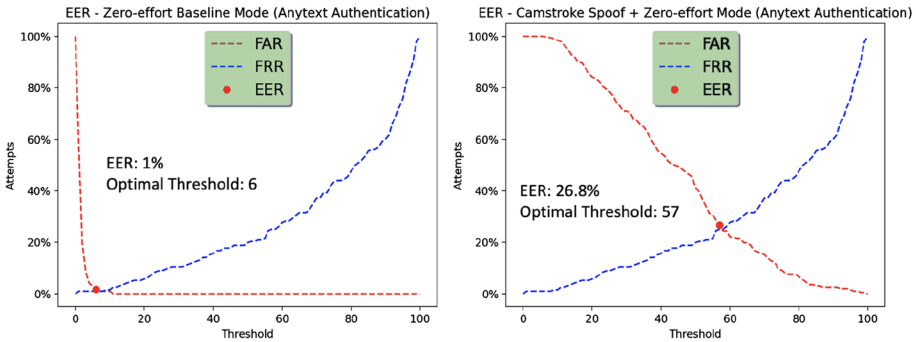

Fig. 17 EER comparison for freetext-mode, before attack (left) and after the attack (right)

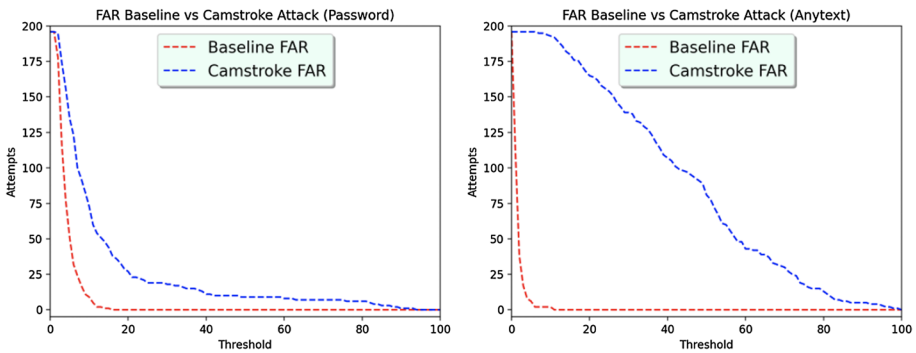

Fig. 18 FAR comparison on password and freetext-mode

FAR value of the authentication system. The comparison between baseline FAR and FRR with FAR and FRR obtained after the attacks have been performed is shown in Figs. 16 and 18. From those figures, we can see that the EER value before and after the attack significantly increases on both Password-mode authentication and Freetext-mode authentication. On Password-mode authentication, the EER value of the system is increased by 349.5% due to the attack. Meanwhile, an EER increase of 2553.5% is reported on Freetext-mode authentication. This is due to the increase in the False Acceptance Rate (FAR) of the authentication system, as can be seen in Fig. 17.

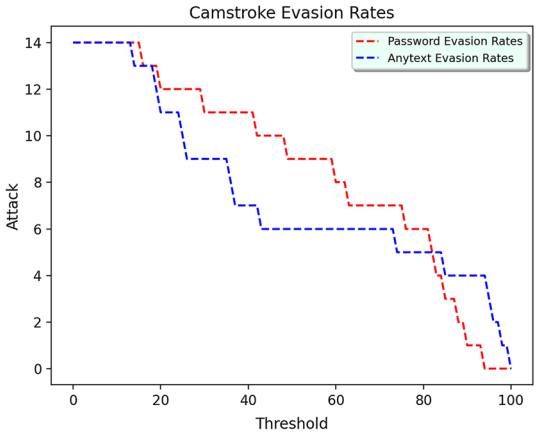

Fig. 19 Evasion rates on KeyTrac authentication

Evasion rate measurement

From 14 subject participants, the evasion rate, which is measured by how many attacks are successful, is also measured and shown in Fig. 19. Note that for the evasion rate measurement, zero-effort attack was not taken into consideration, only spoof attack. Hence, only 14 spoofed typing samples (instead of 196 attack samples) are used in this measurement. With the authentication threshold of 50, the proposed method is able to obtain a 64.3% evasion rate when spoofing Password-mode authentication, and 43% evasion rate when spoofing Freetext-mode authentication. However, we discovered that the proposed methods were not able to achieve a higher evasion rate, particularly on Freetext-mode, because while the proposed method is able to extract *KeyDelay* timing (i.e. the delays between two consecutive characters), it is not able to extract *KeyHold* timing yet (i.e. the delay between a press event and release event of a single keystroke). Meanwhile in KeyTrac, as in many other Keystroke Dynamics authentication services, they generally relied on two typing metrics: *KeyHold* and *KeyDelay*. Hence, as this simulation uses a constant value of KeyHold timing to prevent them from affecting the authentication process, it also affects the ability to imitate the target user's typing pattern on an established authentication service such as KeyTrac. Even though, we believe that this is a considerably acceptable tradeoff between using an actual keylogger where both *KeyHold* and *KeyDelay* can be recorded with high precision but requires active interaction or malware infection, with our proposed method that eliminates those requirements.

Big data application

Collecting and processing a number of screen-record videos consumes both time and resources very extensively. The previous results shows that our proposed method require an approximately 5–7 times the actual duration of the video in order to extract typing patterns and activities in the video. Although we believe that spoofing attack is not a time-sensitive action, meaning that the attacker should not have to commence the attack as soon as the screen-record videos is received, an optimization of the entire process can be applicable by utilizing some of the Big Data approaches.

For example, take a case where an attacker collected screen-record video data from 14 subjects at the same time, and all of them needs to be processed at once, the time

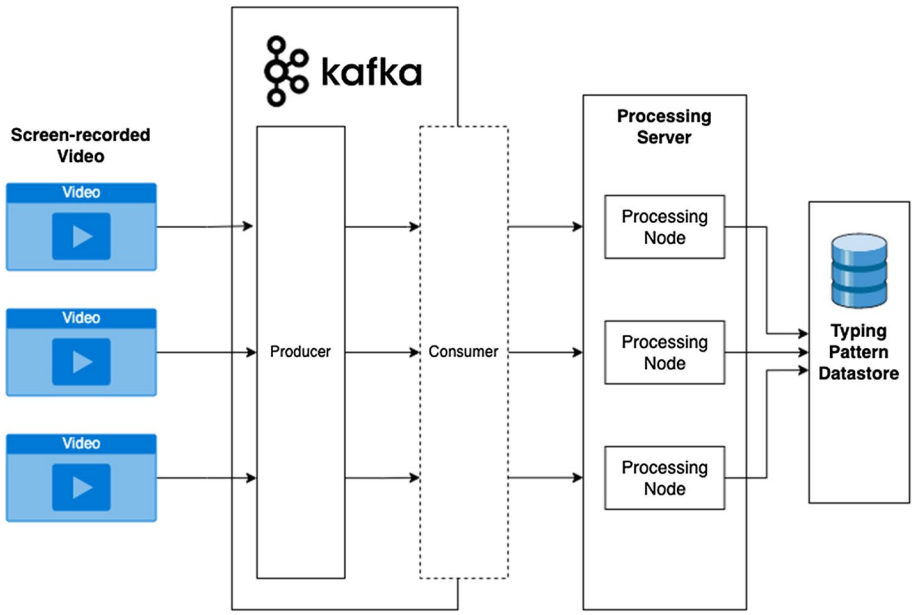

Fig. 20 Big Data pipeline for keystroke extraction

Table 9 Attack specification of some keystroke dynamics spoofing methods

Methods	Attack medium	Target platform	Attack requirement
Malboard [21]	Hardware (USB) Keylogger	Computer	1. Need to install arbitrary keylogger hardware (USB) 2. Require more combinations of keystrokes data collected to achieve higher similarity
Mimicry [30]	Video of Smartphone Typing Activities (Touch-screen)	Smartphone	1. Victim's finger must be visible on the video 2. Attacker needs to train themselves to mimic the victim's typing behaviour, the proposed method only provides visual real-time guidance
Our Method	Screen-recorded Video	Both Computer & Smartphone	1. Victim's screen must be recorded (e.g., through screen sharing activity) 2. Require more combinations of keystrokes data collected to achieve higher similarity

and resources consumption will be very humongous. With our current approach, the videos will have to be queued and processed consecutively. In overall, the attacker is estimated to have to wait 49 min ($30s \times 7 \times 14 video$) to process the 30-s video of the 14 victims. Hence, to optimize the process, a streaming pipeline can be utilized, as shown in Fig. 20. This pipeline can be impactful when a mass-extraction of screen-record videos is needed: i.e., the attacker deployed multiple agents that record the screen of multiple computers at once remotely.

The parallelism of pattern extraction processes can either be done in a single but powerful server, hence reducing the time required to process the videos; or on a separate but identical standard-usage server, thereby distributing the resource-load while altogether still reducing the time consumption. Furthermore, our proposed method does not have any constraint or limitation with regard to be

Table 10 Capability and limitations of some keystroke dynamics spoofing methods

Methods	Attacker interaction	Remote exploitation	Detection likelihood	Other limitations
Malboard [21]	*Required*, attacker must install hardware keylogger into victim's keyboard	No	*Likely*, the attached hardware keylogger is easier to be noticed	The attack requires Internet connection to transfer the collected keystroke behaviours
Mimicry [30]	*Required*, attacker must be able to record the victim's finger and the smartphone's screen while the victim is typing	No	*Likely*, the attacker must be in close proximity to the victim to record their typing activity on the smartphone	The attack is not automated, the proposed method only provides visual guidance.
Our Method	*Semi-required*, attacker could passively obtain the screen-recorded video via the victim's screen-sharing activity	Yes	*Unlikely*, the victim is less-likely to notice when their screen is being recorded remotely (e.g., via the screen-sharing activity)	Evasion Rates (ER) is lower, and only lowercase characters are supported (as of now)

process-paralleled, so the screen-recorded video and the typing pattern that we aim to extract can be processed independently, thus could very much be applied within a Big Data ecosystem.

Method comparison and limitations

Some research have been previously conducted to propose nefarious attacks against Keystroke Dynamics, especially on impersonation, mimicry, and spoofing attacks. In hindsight, to be able to spoof keystroke dynamics authentication systems successfully, an attacker must be able to type with style that is similar to the victim's. Table 9 shows different approaches that are proposed to help an attacker in collecting someone's typing behavior and mimic them to spoof a keystroke dynamics authentication. Although some of those precedent methods had achieved rather high evasion rates, or even introduce a clever and persistent threat to the victims, some gaps might still be exist in terms of whether the attack can be remotely conducted, whether the attack is likely to be detected, and if an interaction with the victim is required. The general characteristics along with the limitations of their proposed methods, compared with ours, are summarized and displayed in Table 10.

Furthermore, we believe that our process of extracting the victim's typing delays is not applicable in a situation where the typed phrases (e.g., password)are not visible on the screen, or if they are masked into different symbols (i.e., * or · symbols) other than the characters themselves. To be able to similarly mimic the typing delays between each character, our method needs to extract information about what character is being typed and how long is the delay until the next character is typed. Hence, when the characters are masked into different symbols or not visible at all on the screen, our method will not be able to gain the necessary information to recreate the synthetic typing patterns. This is a drawback that we think is reasonably acceptable as it would help extract and mimic each of the targeted user's typing patterns with quite a high similarity without generalizing the pattern. However, in a situation where the typed characters are masked or not visible on the screen, one might want to resort back to generic typing datasets that are available on many public sources [36–38], and identify which of them are similar enough to the typing pattern of the targeted user to be used for the spoofing attack.

Conclusion and future work

This paper proposes a novel attack method against keystroke dynamics authentication. Our proposed approach combines the capability of computer vision and statistical analysis to extract typing patterns that belongs to a user from video frames. By using a screen-recorded video, someone can achieve a staggering similarity in certain keystroke timing patterns as if they used a keylogger, shown by the statistical tests that indicate there are mostly no significant mean differences between them. Moreover, by relying on videos, especially screen-recorded video, the proposed method eliminates the need for any external hardware or modifications on the computer's victims; hence the attack can be performed remotely and more quietly. A series of experiments consisting of various text-typing scenarios were conducted with 14 participating subjects. The experiment result shows that our proposed approach is able to extract the delays between one keystroke to another with close similarity to the actual delays. Finally, from the attack simulation against KeyTrac, our proposed method obtained evasion rate as high as 64.3% for Password-mode and 43% for Freetext-mode, indicating that although they are still a little bit low, the risk our proposed attack yields must be carefully considered as the attack can be devastating if successful, given the less limitations and less-complexity of the attack as well as the fact that ours can be conducted remotely. In future work, we plan to:

1. Develop a technique to predict accurate delays from non-typed characters, i.e., characters that don't appear on the video frames.
2. Improve the text-cursor tracking and character recognition models to for advanced conditions, e.g., different fonts, scrolling-screens, etc.
3. Develop an algorithm to infer and capture not only KeyDelay timing, but also KeyHolds from a screen-recorded video.
4. Benchmark the proposed attack against other keystroke dynamics authentication services (e.g., Behaviosec, TypingDNA, etc.) and measure the evasion rate of the proposed attack.

Hopefully, this research could also demonstrate the susceptibility of Keystroke Dynamics to a behavioral-information leakage and trigger the urgency for the computer security researcher and community regarding the potential exploits against Keystroke Dynamic authentication so that its security can be taken into more attention and improved in future advances.

Author contributions
CS and AC are the authors in this paper. Both authors read and approved the final manuscript.

Authors' information
CS is a Cybersecurity lecturer in Bina Nusantara University, Indonesia. CS received his Bachelor's degree in Cybersecurity and a master's in Data Science from Bina Nusantara University ID. His research interest includes endpoint and system security, advanced persistent threat (APT) and novel attack vectors, as well as malware development and behavior-based exploitations. Having a solid background in conducting vulnerability assessment, penetration testing, and a deep understanding of Web and Mobile security architecture, CS is also interested in Deep Learning and Artificial Intelligence domains, including computer vision, all to combine and expand his Cybersecurity research and techniques. Andry is a Computer Science Senior lecturer in Bina Nusantara University Indonesia. Andry received his Ph.D. degree in Computer Science from The University of Nottingham UK, a master degree in Business Management from BINUS Businesss School ID, and a bachelor degree in Computer Science from BINUS University ID. His research is in agent architecture and Machine (and Deep) Learning. His work mainly on how to model an agent that has capability to sense and perceive the

environment and react based on the perceived data in addition to the ability of building a social relationship with the user overtime. In addition, Andry is also interested in serious game and gamification design.

Funding
Not applicable.

Availability of data and materials
Not applicable.

Declarations

Ethics approval and consent to participate
Not applicable.

Consent for publication
Not applicable.

Competing interests
The authors declare that they have no competing interests.

Received: 8 February 2022 Accepted: 20 October 2022
Published online: 21 November 2022

References

1. De Luca A, Denzel M, Hussmann H. Look into my eyes! can you guess my password? In: Proceedings of the 5th Symposium on Usable Privacy and Security; 2009. p. 1–12.
2. Lashkari AH, Farmand S, Zakaria D, Bin O, Saleh D et al. Shoulder surfing attack in graphical password authentication. arXiv preprint arXiv:0912.0951 2009.
3. Shukla D, Phoha VV. Stealing passwords by observing hands movement. IEEE Trans Inform For Secur. 2019;14(12):3086–101.
4. Sun Y, Upadhyaya S. Synthetic forgery attack against continuous keystroke authentication systems. In: 2018 27th International Conference on Computer Communication and Networks (ICCCN), 2018. p. 1–7.
5. Mhenni A, Migdal D, Cherrier E, Rosenberger C, Amara NEB. Vulnerability of adaptive strategies of keystroke dynamics based authentication against different attack types. In: 2019 International Conference on Cyberworlds (CW), 2019. p. 274–8.
6. González N, Calot EP, Ierache JS, Hasperué W. The reverse problem of keystroke dynamics: Guessing typed text with keystroke timings only. In: 2021 International Conference on Electrical, Computer and Energy Technologies (ICECET), 2021. p. 1–6.
7. Balagani K, Cardaioli M, Conti M, Gasti P, Georgiev M, Gurtler T, Lain D, Miller C, Molas K, Samarin N, et al. Pilot: Password and pin information leakage from obfuscated typing videos. J Computer Secur. 2019;27(4):405–25.
8. Balagani KS, Conti M, Gasti P, Georgiev M, Gurtler T, Lain D, Miller C, Molas K, Samarin N, Saraci E, *et al.* Silk-tv: Secret information leakage from keystroke timing videos. In: European Symposium on Research in Computer Security. New York: Springer; 2018. p. 263–80.
9. Teh PS, Teoh ABJ, Yue S. A survey of keystroke dynamics biometrics. The Scientific World Journal (2013).
10. Obaidat MS. A verification methodology for computer systems users. In: Proceedings of the 1995 ACM Symposium on Applied Computing, 1995. p. 258–62.
11. Giuffrida C, Majdanik K, Conti M, Bos H. I sensed it was you: authenticating mobile users with sensor-enhanced keystroke dynamics. In: International Conference on Detection of Intrusions and Malware, and Vulnerability Assessment. New York: Springer; 2014. p. 92–111.
12. Messerman A, Mustafić T, Camtepe SA, Albayrak S. Continuous and non-intrusive identity verification in real-time environments based on free-text keystroke dynamics. In: 2011 International Joint Conference on Biometrics (IJCB), IEEE; 2011. p. 1–8.
13. Xi K, Tang Y, Hu J. Correlation keystroke verification scheme for user access control in cloud computing environment. Computer J. 2011;54(10):1632–44.
14. Stewart JC, Monaco JV, Cha S-H, Tappert CC. An investigation of keystroke and stylometry traits for authenticating online test takers. In: 2011 International Joint Conference on Biometrics (IJCB), IEEE; 2011. p. 1–7.
15. Leberknight CS, Widmeyer GR, Recce ML. An investigation into the efficacy of keystroke analysis for perimeter defense and facility access. In: 2008 IEEE Conference on Technologies for Homeland Security, IEEE; 2008. p. 345–50.
16. Ogihara A, Matsumura H, Shiozaki A. Biometric verification using keystroke motion and key press timing for atm user authentication. In: 2006 International Symposium on Intelligent Signal Processing and Communications, IEEE; 2006. p. 223–6.
17. Mandujano S, Soto R. Deterring password sharing: User authentication via fuzzy c-means clustering applied to keystroke biometric data. In: Proceedings of the Fifth Mexican International Conference in Computer Science, 2004. ENC 2004., IEEE; 2004. p. 181–7.
18. Alamri EK, Alnajim AM, Alsuhibany SA. Investigation of using captcha keystroke dynamics to enhance the prevention of phishing attacks. Future Internet. 2022;14(3):82. https://doi.org/10.3390/fi14030082.
19. Sultanov A, Kogos K. Insider threat detection based on stress recognition using keystroke dynamics. CoRR **abs/2005.02862** (2020). arXiv:2005.02862.

20. Maalej A, Kallel I. Does keystroke dynamics tell us about emotions? a systematic literature review and dataset construction. In: 2020 16th International Conference on Intelligent Environments (IE). IEEE; 2020. p. 60–7.
21. Farhi N, Nissim N, Elovici Y. Malboard: A novel user keystroke impersonation attack and trusted detection framework based on side-channel analysis. Computers Security. 2019;85:240–69.
22. Joyce R, Gupta G. Identity authentication based on keystroke latencies. Commun ACM. 1990;33(2):168–76.
23. Araújo LC, Sucupira LH, Lizarraga MG, Ling LL, Yabu-Uti JBT. User authentication through typing biometrics features. IEEE Trans Signal Process. 2005;53(2):851–5.
24. Bleha S, Slivinsky C, Hussien B. Computer-access security systems using keystroke dynamics. IEEE Trans Pattern Anal Mach Intell. 1990;12(12):1217–22.
25. Kang P, Hwang S-s, Cho S. Continual retraining of keystroke dynamics based authenticator. In: International Conference on Biometrics, Springer; 2007:1203–1211.
26. Jain AK, Ross AA, Nandakumar K. Introduction to Biometrics. New York: Springer; 2011.
27. Kulakis G, Olson K, Doremus J, Geiger S, Monaco JV. Obtaining receiver operating characteristic curves from commercial biometric systems, 2016.
28. Chirita S. NY DMV approves Keystroke Analysis as a validation method, 2021. https://blog.typingdna.com/dmv-approves-keystroke-analysis/.
29. Sloan T. European Banking Authority identifies approved methods for Biometrics, 2019. https://www.paymentsjournal.com/european-banking-authority-identifies-approved-methods-for-biometrics/.
30. Khan H, Hengartner U, Vogel D. Mimicry attacks on smartphone keystroke authentication. ACM Trans Privacy Security (TOPS). 2020;23(1):1–34.
31. Chen Y, Li T, Zhang R, Zhang Y, Hedgpeth T. Eyetell: Video-assisted touchscreen keystroke inference from eye movements. In: 2018 IEEE Symposium on Security and Privacy (SP), IEEE; 2018. p. 144–60.
32. Bradski G. The OpenCV Library. Dr. Dobb's Journal of Software Tools, 2000.
33. Phinney T. Point size and the EM Square: Not what people think, 2012. http://www.thomasphinney.com/2011/03/point-size/.
34. He K, Zhang X, Ren S, Sun J. Deep residual learning for image recognition. In: Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2016. p. 770–8.
35. Rey D, Neuhäuser M. In: Lovric, M. (ed.) Wilcoxon-Signed-Rank Test, 2011:1658–1659. Springer, Berlin, Heidelberg.
36. Killourhy KS, Maxion RA. Comparing anomaly-detection algorithms for keystroke dynamics. In: 2009 IEEE/IFIP International Conference on Dependable Systems & Networks, IEEE; 2009. p. 125–34.
37. Killourhy KS, Maxion RA. Free vs. transcribed text for keystroke-dynamics evaluations. In: Proceedings of the 2012 Workshop on Learning from Authoritative Security Experiment Results, 2012. p. 1–8.
38. Banerjee R, Feng S, Kang JS, Choi Y. Keystroke patterns as prosody in digital writings: A case study with deceptive reviews and essays. In: Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP), 2014:1469–1473. Association for Computational Linguistics, Doha, Qatar. http://www.aclweb.org/anthology/D14-1155.

Publisher's Note

Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.

Submit your manuscript to a SpringerOpen journal and benefit from:

- ► Convenient online submission
- ► Rigorous peer review
- ► Open access: articles freely available online
- ► High visibility within the field
- ► Retaining the copyright to your article

Submit your next manuscript at ► springeropen.com