On the Long-Term Effects of Continuous Keystroke Authentication: Keeping User Frustration Low through Behavior Adaptation

JUN HO HUH, Samsung Research, Republic of Korea
SUNGSU KWAG, Samsung Research, Republic of Korea
ILJOO KIM, Samsung Research, Republic of Korea
ALEXANDR POPOV, Samsung R&D Institute Ukraine, Ukraine
YOUNGHAN PARK, Moloco Inc., United States
GEUMHWAN CHO, Sungkyunkwan University, Republic of Korea
JUWON LEE, Samsung Research, Republic of Korea
HYOUNGSHICK KIM, Sungkyunkwan University, Republic of Korea
CHOONG-HOON LEE, Samsung Research, Republic of Korea

One of the main challenges in deploying a keystroke dynamics-based continuous authentication scheme on smartphones is ensuring low error rates over time. Unstable false rejection rates (FRRs) would lead to frequent phone locks during long-term use, and deteriorating attack detection rates would jeopardize its security benefits. The fact that it is undesirable to train complex deep learning models directly on smartphones or send private sensor data to servers for training present unique deployment constraints, requiring *on-device* solutions that can be trained fully on smartphones.

To improve authentication accuracy while satisfying such real-world deployment constraints, we propose two novel feature engineering techniques: (1) computation of pair-wise correlations between accelerometer and gyroscope sensor values, and (2) on-device feature extraction technique to compute dynamic time warping (DTW) distance measurements between autoencoder inputs and outputs via transfer-learning. Using those two feature sets in an ensemble blender, we achieved 6.4 percent equal error rate (EER) in a public dataset. In comparison, blending two state-of-the-art solutions achieved 14.1 percent EER in the same test settings. Our real-world dataset evaluation showed *increasing* FRRs (user frustration) over two months; however, through periodic model retraining, we were able to maintain average FRRs around 2.5 percent while keeping attack detection rates around 89 percent. The proposed solution has been deployed in the latest Samsung Galaxy smartphone series to protect secure workspace through continuous authentication.

CCS Concepts: • **Security and privacy** → **Usability in security and privacy**; **Biometrics**.

Additional Key Words and Phrases: keystroke dynamics, smartphone authentication, real-world dataset

ACM Reference Format:
Jun Ho Huh, Sungsu Kwag, Iljoo Kim, Alexandr Popov, Younghan Park, Geumhwan Cho, Juwon Lee, Hyoungshick Kim, and Choong-Hoon Lee. 2023. On the Long-Term Effects of Continuous Keystroke Authentication: Keeping User Frustration

Authors' addresses: Jun Ho Huh, junho.huh@samsung.com, Samsung Research, Seoul, Republic of Korea; Sungsu Kwag, sungsu.kwag@samsung.com, Samsung Research, Seoul, Republic of Korea; Iljoo Kim, ij00.kim@samsung.com, Samsung Research, Seoul, Republic of Korea; Alexandr Popov, ol.popov@samsung.com, Samsung R&D Institute Ukraine, Kiev, Ukraine; Younghan Park, younghanpark@gmail.com, Moloco Inc., Redwood City, CA, United States; Geumhwan Cho, geumhwan@skku.edu, Sungkyunkwan University, Suwon, Republic of Korea; Juwon Lee, juwon14.lee@samsung.com, Samsung Research, Seoul, Republic of Korea; Hyoungshick Kim, hyoung@skku.edu, Sungkyunkwan University, Suwon, Republic of Korea; Choong-Hoon Lee, choonghoon.lee@samsung.com, Samsung Research, Seoul, Republic of Korea.

2474-9567/2023/6-ART58 $15.00
https://doi.org/10.1145/3596236

Low through Behavior Adaptation. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.* 7, 2, Article 58 (June 2023), 32 pages. https://doi.org/10.1145/3596236

1 INTRODUCTION

Unlike traditional authentication systems that require users to authenticate once during initial login, *continuous authentication* implies that users' authenticity is checked continuously throughout active sessions [41, 49]. This notion has been studied extensively in the context of strengthening smartphone security [32] where many users solely rely on explicit authentication schemes like patterns and PINs to protect their devices: prior research has shown that users often select easy-to-remember patterns [13, 48] and PINs [28] that are also easy to guess, and vulnerable to a probabilistic model based guessing techniques [6], dictionary attacks [13], and smudge attacks that look for guessing hints from fingerprint trails [10]. To mitigate such risks associated with bypassing explicit authentication, this paper proposes a continuous authentication system that checks users' daily typing behaviors to detect anomalous behaviors, and prevent further access.

Phones can be locked immediately if an anomalous behavior is detected, and request for a second-factor verification through a different method (e.g., a biometric scheme) to regain access. A more relaxed access control policy, for example, could deny access to security-critical apps (e.g., banking apps) until normal behaviors are detected again. Alternatively, a series of continuous authentication scores can be used to evaluate security risks [42], and require second-factor verification to access certain apps when the measured risks are high. Some European banks have started accepting keystroke dynamics (typing behavior) based authentication as one of the two-factor schemes for allowing user access and money transfer [18].

On smartphones, users' touchscreen keyboard typing behaviors – hereafter referred to as "keystroke dynamics" – would provide intuitive and implicit means to verify users' authenticity. A keystroke dynamics-based authentication scheme would work by training a user's keystroke behaviors, and using a trained classifier at run-time to continuously verify the user's authenticity while typing (e.g., sending an email). According to [25], a frequent source of annoyance for people using implicit authentication is false rejection rates (FRRs), which represent error rates associated with users' own actions being rejected (affects system usability); they also highlight that attack success rates, equivalent to false acceptance rates (FARs), are major security concerns for users. Our initial real-world error rate analysis demonstrated that FRRs (associated with unwanted phone lockouts) may elevate over time, and highlighted the importance of keeping FRRs consistently low throughout long-term use. FARs also need to be maintained to guarantee long-term security benefits. However, previous literature have often overlooked such long-term accuracy challenges.

Recent research [1, 2, 9] has explored deep learning techniques to automate feature extraction process, and build solutions that are more robust to changing behaviors and new contexts. In real-world deployment settings, however, it is undesirable to train complex deep learning models directly on smartphones considering computational overheads. Our interview study revealed that training them on servers is also undesirable due to privacy issues associated with sharing sensor data. Prior work exploring deep learning techniques [1, 2, 9], however, have not considered these constraints. This paper is motivated by the need to develop an *on-device* solution that can maintain low error rates over time.

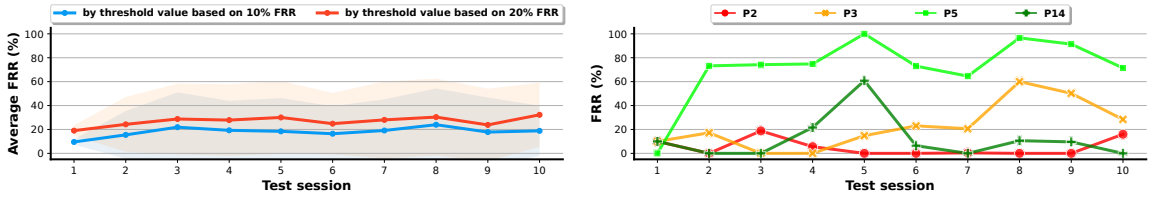
We propose a keystroke dynamics-based authentication solution (KedyAuth) that is capable of achieving an average equal error rate (EER) of 6.6 percent on the widely referenced "HMOG" (hand movement, orientation, and grasp) dataset [9, 38, 41]. An EER is measured based on a threshold that achieves the same FAR and FRR on a given authentication system. To avoid training deep learning models on smartphones, we propose a novel transfer learning method that loads a pre-trained autoencoder to first measure the differences between original input (sensor values) and reconstructed (decoder) output, and uses these distance measurements as the classification features. KedyAuth also uses novel correlation features extracted from computing Pearson's correlation coefficients between accelerometer and gyroscope sensor values. In comparison, combining the

extensively studied HMOG features [41], and temporal and spatial features [8] achieved 14.1 percent EER in the same test settings. To directly compare performance, we select existing solutions that can also be trained fully on smartphones. To study long-term effects, we collected real-world keystroke data over two months by handing out smartphones, and asking participants to use them as their primary devices. We tested the best-performing blender on this dataset, showing that FRRs can be inconsistent, and users may experience more keystrokes being rejected over time. To mitigate this usability issue, we experimented with several model retraining techniques. Our contributions are summarized below:

- **Smartphone deployment guidelines.** Through an interview study (N=20) and a technical workshop conducted at a large smartphone company, we identified highly concerning threats that involve attackers typing text (e.g., sending a phishing email on a stolen phone), and integral deployment constraints (e.g., the need to train lightweight classifiers fully on devices) that should be considered upon deploying a machine-learning based authentication solution on smartphones.
- **Novel on-device feature engineering techniques.** We propose (1) correlation features that capture keystroke dynamics and characteristics in higher dimensional space constructed by multiple (gyroscope and accelerometer) sensors, and outperform state-of-the-art methods, and (2) an on-device feature extraction technique that measures dynamic time warping (DTW) distance between sensor inputs and outputs reconstructed through a pre-trained (common) autoencoder, and uses these distance measurements to distinguish users; this technique achieves 8.0 percent EER, and demonstrates top performance on the HMOG dataset.
- **2-month real-world touchscreen use data collection and long-term accuracy evaluations.** We demonstrate that model retraining is necessary to maintain low error rates over time: a retraining technique that uses all stacked samples available can maintain FRRs around 2.5 percent while keeping attack detection rates at 89 percent with a default threshold. To the best of our knowledge, we are the first group to thoroughly analyze long-term FRR inconsistency issues based on a real-world dataset (N=40), and propose efficient on-device retraining schemes for typing behavior adaptation.
- **Real-world deployment.** Unlike existing solutions that focus on demonstrating one-time error rates, KedyAuth has been designed to (1) maintain low error rates over long-term use, and (2) facilitate light and fast on-device feature extraction and model retraining. Each retraining would take about 6.8 seconds to complete. Lowering thresholds to guarantee negligible FRRs would result in about 73 percent of unauthorized phone access being detected – this is still a significant improvement from zero continuous authentication. KedyAuth has been deployed on the latest Samsung Galaxy series to facilitate continuous protection of secure workspace.
- **Retraining overhead evaluations.** Frequent model retraining may quickly drain phone battery, and affect users' typing experiences (e.g., introducing keyboard lags). Our overhead measurement study (N=9) shows KedyAuth – configured to continuously authenticate every keystroke, and retrain the full ensemble blender every 500 new samples – would consume only about 0.45 percent of battery per day for average users, and 127 megabytes of additional memory when users type. Most participants did not notice any keyboard lag while typing; those who did experience one or two lags explained they were weakly or barely perceptible.

2 THREATS AND REQUIREMENTS

In this section, we present the long-term error rate problem through the analysis of a two-month real-world keystroke dataset, and motivate the need to build an adaptive system and support model retraining. We also explain the first interview study conducted to understand users' smartphone security concerns, and privacy



(a) Average FRRs

(b) FRR plots for problematic users

Fig. 1. (a) The average FRRs of each test session measured over 20 participants; blue and red lines represent thresholds that guarantee 10 and 20 percent FRRs on the first test session; the bands represent standard deviations; (b) FRR plots based on the 10 percent FRR threshold for P2, P3, P5 and P14.

concerns they might have in using KedyAuth. Based on those observations, we provide a summary of design guidelines for building and deploying a practical KedyAuth system.

2.1 Inconsistency in Long-term Error Rates

Khan et al. [25] explain the primary usability concern people have about using implicit authentication is the error rates associated with falsely being rejected from their own devices (FRRs). People also worry about attackers bypassing implicit authentication (FARs) and attack detection delays, highlighting that keeping both error rates low and consistent are important system requirements.

Prior work on continuous authentication [11] revealed that users' interactive behaviors may change over time – potentially leading to elevations in FRRs (user frustration) during long-term use. To investigate the severity of this long-term FRR inconsistency problem in the context of keystroke dynamics-based authentication, we implemented the well-established (and validated) HMOG-GR feature set [41]: micro phone movements and orientation changes associated with key tapping events are modeled through the use of accelerometer and gyroscope sensor values. We measured its long-term performance through a real-world keystroke dataset collected over a two-month period (see Section 4.1 for the dataset details). From 40 participants, we selected 20 participants with at least 7,500 keystroke samples for this evaluation – we measured FRRs based on 10 temporal test sessions, each session consisting of 500 samples. For each participant, we trained a user-specific one-class support vector machine (SVM) classifier using the first 2,500 samples, fixed an anomaly/outlier detection threshold value that guarantees roughly 10 percent FRR on the first test session samples, and used that threshold to measure the FRRs for the 9 subsequent test sessions. We repeated the same evaluation with another threshold that guarantees about 20 percent FRR on the first test session. Users' own keystrokes that score less than the threshold are counted as false rejections. Note, the average (one-class classification) EER over the 20 participants for the first test session (500 samples) was 19.8 percent ($\sigma = 12.0$).

Figure 1a shows the average FRRs for each test session computed over 20 participants. The blue and red lines represent the FRRs measured using the 10 percent and 20 percent thresholds, respectively. Both graphs show an increasing trend, and the standard deviation bands indicate concerning (unacceptably high) levels of FRRs for some participants. We select three problematic participants, and show their FRR trends and variations in Figure 1b. P3 would experience significant FRR elevations in the 8th (60 percent) and 9th (50 percent) test sessions. P14 would experience about 60 percent FRRs in the 5th session. P5 would experience most of his or her own keystrokes being rejected from the second session. If the original classifiers are used without any update, many users will experience inconsistent system behavior, and encounter an unacceptably large number of their own keystrokes being rejected. For comparison, we also show a consistent FRR trend through P2.

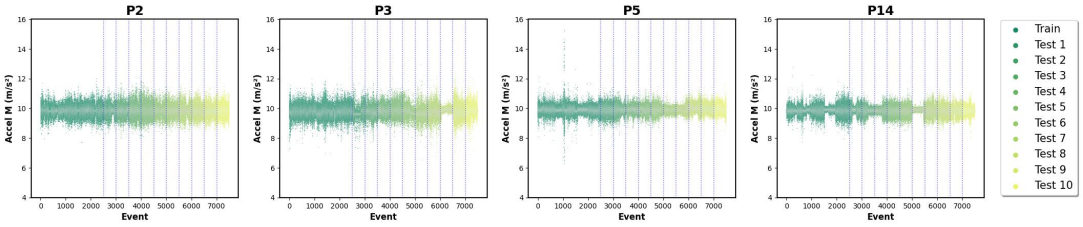


Fig. 2. Accelerometer magnitude (M) values for P2, P3, P5, and P14. M is the square root of the sum of the squares of X, Y, Z values. Vertical dotted lines divide test sessions. First 2,500 samples represent the train set.

To provide more insight into these varying trends, we study how the raw sensor values (corresponding to key taps) may have changed over time. Graphs in Figure 2 show the raw accelerometer magnitude (M) values across the 10 test sessions for those four participants. M is the square root of the sum of the squares of X, Y, Z values. These raw values tend to show more consistent distributions and ranges for P2. For P3, P5, and P14, however, the raw values in several test sessions tend to show vastly different distributions and ranges compared to the train samples (up to 2,500). Since the HMOG-GR features compute average and standard deviations, such significant shifts in raw value distributions would also affect the features. Considering that raw accelerometer values (as well as the computed HMOG-GR) are impacted by how people hold their phones, and exert and resist force during tap events, these observations suggest the participants' typing behaviors and characteristics may have changed in later sessions, or they may have adopted new body poses (e.g., walking or leaning) over time. In consequence, classifiers trained with the initial 2,500 samples may become less effective over time, as they are no longer suitable for classifying samples in subsequent test sessions that represent vastly different typing behaviors or new body poses. To maintain the accuracy during long-term use, classifiers must be periodically retrained with new keystroke data.

One security risk of an adaptive system and frequent model retraining is that FARs could become unstable over time: this is because including new train samples that represent different typing behaviors and poses may affect users' classification boundaries, and their classifiers could be updated in a way that more attack samples are accepted as users' own samples. Such long-term FAR inconsistency risks sit uneasily with users' security expectations, and could seriously jeopardize the security benefits. We highlight the need to design an adaptive system that is capable of keeping long-term FARs consistent, and providing the same level of security guarantees (protections) throughout long-term use.

2.2 Security and Privacy Concerns

We conducted an interview study with 20 participants to investigate (1) concerning unauthorized phone access threats that involve the use of keyboards, and (2) users' privacy perceptions toward the collection and use of sensitive keystroke data to enable KedyAuth. In addition, we arranged a workshop with engineers from a large smartphone vendor to investigate real-world deployment constraints that need to be considered. Based on the interview responses and workshop outcomes, we identify key security and usability requirements for KedyAuth deployment.

2.2.1 Methodology. We conducted a semi-structured interview study to understand concerns users might have with their phones and apps being used by unauthorized parties, and identify threats that specifically involve the use of touchscreen keyboards. We also asked questions about security and usability expectations users might have with respect to using KedyAuth. We recruited 20 participants who are aged 18 years or older by posting advertisements on online notice boards at the university as well as selectively recruiting people from local

communities based on age and work experiences to reduce bias in demographics. Two moderators ensured the interview questions were consistently understood by the participants. Each study session took about 30 minutes to complete, and the participants were compensated with a gift card equivalent of 10 USD in local currency. All interviews were recorded and transcribed.

As for all open-ended questions, two researchers separately coded each interview data and created a common codebook through iterative discussions. We applied structural coding techniques [27] to identify six topic codes. After resolving coding disagreements, we achieved an inter-coder agreement of 90% Cohen's Kappa [20]. We informed the participants that participation is voluntary and confidential, and they have the right to terminate the study without any penalty. We asked for their permission to audio-record entire interview sessions. The ethical perspective of this study has been approved by the university's IRB.

During the first part of the interview, we asked the participants to imagine three separate scenarios in which a family member, a co-worker, and a stranger (threat actors) manages to gain physical access to their phones, and knowing the screen lock secret, unlocks their phones, and uses any app installed. After giving the participants some time to think through different possible attack scenarios, we asked them to provide a list of apps they would be concerned about (Q1). As the next step, we went through each app mentioned, and asked the participants to explain specific attack scenarios (e.g., "a stranger tries to send money to arbitrary accounts using my banking app") that they are concerned with.

Before starting the second part of the interview, we explained the basic concepts of KedyAuth (including how continuous keystroke verification is operated) and how it can protect users from the threats mentioned in the first part of the study. We then played a short demo video to visually demonstrate how KedyAuth checks all keystrokes to prevent strangers from continuously using apps installed on stolen devices – showing that strangers, after typing a few words, eventually get locked out. After watching this video, we asked *"KedyAuth would have to collect your keystroke data in the form of accelerometer, gyroscope, and touch sensor data, and store them either on your phone or on a remote server for processing. Would you be willing to share your keystroke information with a phone vendor like Apple or Samsung to enable KedyAuth and protect your phone?"* (Q2). For those who said they were not willing to share this data with a phone vendor, we asked a follow-up question *"Would you be willing to store that information just locally on your phone to enable KedyAuth?"* The reason we explained the basic KedyAuth concepts prior to asking those questions was to investigate whether the participants are willing to share their keystroke data fully understanding its security benefits, and answer those questions in the correct and specific context of KedyAuth features. Before conducting the interview, we conducted a pilot study with three participants, and used their feedback to fix study protocol issues.

To understand system deployment constraints, we also arranged a workshop with five security engineers and one project manager from a large smartphone vendor who is currently developing user behavior-based authentication solutions. During this workshop, we presented our solution – explaining key concepts related to extracting features based on users' touch, accelerometer, and gyroscope sensor data, retraining classifiers, and continuously verifying users' keystrokes – and received technical feedback on deployment constraints that should be considered. We summarize these constraints in Section 2.3.

2.2.2 Demographics. We interviewed a total of 20 participants. 11 out of 20 were females, and the average age was 37.8 (σ = 10.9). 15 participants had a college or university degree. The participants said, on average, they spend about 6 hours per day on their phones. 16 participants said they store sensitive or confidential information on their phones. 9 different occupations were reported. The demographics of those 20 participants are presented in Table 1. We compared the age, gender, and education distributions against the US smartphone population reported in [45], performing Fisher's exact tests to show that there is no significant difference in age ($p = 0.15$), gender ($p = 0.66$), and education ($p = 0.65$) distributions.

Table 1. Demographics of the interview study participants.

Gender		
Female	11	(55%)
Male	9	(45%)
Age		
18–29	5	(25%)
30–39	8	(40%)
40–49	3	(15%)
50–59	3	(15%)
60 and more	1	(5%)
Education		
Less than high school	0	(0%)
High school	4	(20%)
College/University	15	(75%)
Master's/Doctoral degree	1	(5%)
Other	0	(0%)
Occupation		
Arts, Design, Entertainment, Sports, and Media Occupations	3	(15%)
Community and Social Service Occupations	3	(15%)
Computer and Mathematical Occupations	2	(10%)
Education, Training, and Library Occupations	1	(5%)
Office and Administrative Support Occupations	4	(20%)
Production Occupations	1	(5%)
Transportation and Materials Moving Occupations	1	(5%)
Students	2	(10%)
Unemployed/Retired/Disabled	3	(15%)

Table 2. Security-concerning apps that were mentioned by the participants. "*" indicates apps for which one of the mentioned attack scenarios involve the use of keyboards. We count the number of times each threat actor (family member, co-worker, and stranger) was mentioned while describing attack scenarios involving those apps.

App category	Family member	Co-worker	Stranger
Messenger*	12	17	19
Miscellaneous	0	7	19
Phone*	2	5	19
Photo gallery	4	14	18
Banking*	3	10	17
Social media*	3	5	11
Finance (stock trading, credit card)*	1	6	9
Email*	1	1	8

2.2.3 Security Concerns with Installed Apps and Attack Scenarios. Based on the responses to the first question (Q1), we identified app-specific attack scenarios that would involve attackers typing text. All attack scenarios we consider involve an attacker first compromising a victim's screen lock PIN or pattern (e.g., through guessing attacks [10, 13, 28, 48]). When the victim leaves her phone unattended, the attacker unlocks the phone with the compromised PIN or pattern, and uses sensitive apps such as email apps or banking apps with malicious intentions.

Table 2 presents a summary of the apps that the participants were concerned about (Q1), and counts the number of times each threat actor (family member, co-worker, and stranger) was mentioned in a specific concerning attack scenario. "Messenger" app was the most frequently mentioned app. 12 out of 20 participants explained they would be concerned about their family members searching for private or confidential conversations. 2 participants said they are concerned with family members looking at their phone application logs and searching through phone numbers. 17 participants said they do not want their co-workers to look at their messenger conversation logs; 3 participants also said they would be concerned with co-workers sending messages with malicious intentions to their contacts. One participant was concerned about co-workers potentially sending money through his or her messenger app. Another participant was concerned about his or her co-workers using banking or finance apps to perform unauthorized financial transactions.

Everyone except one participant mentioned that they do not want strangers to access any application. 6 participants were specifically concerned with strangers using messenger and email apps to engage in criminal activities such as sending phishing messages or emails to their contacts. 5 participants expressed their concerns about strangers possibly using banking or financial apps to perform unauthorized transactions and withdraw money. 3 participants mentioned that they would be concerned with strangers posting embarrassing content using their social media accounts. The primary goal of KedyAuth is to accurately detect such unauthorized use while attackers type on touchscreen keyboards to perform attacks.

2.2.4 Keystroke Data Storage. In response to Q2, 12 participants expressed privacy concerns with their keystroke data being sent over to a server for processing. 7 participants mentioned they are unwilling to share their touch sensor data with a remote server. Two participants were specifically concerned about the possibility of their passwords being compromised through data sharing. P2 mentioned:

> *"My touch data should not be shared with servers.. I'm worried that my touch sensor data might reveal personal information and secrets like passwords."* (P2)

As for storing keystroke information locally on users' smartphones, everyone except one participant said they have no concern with keystroke data being stored and processed locally. One tech-savvy participant shared a performance-related comment, explaining that the attack detection accuracy could be compromised if data are just stored and processed locally.

2.3 Design Guidelines

Based on the long-term error rate analysis, interview results, and conversations we had with the engineers, we summarize the key design guidelines that should be considered while building and deploying KedyAuth:

(1) **Consistent performance.** Without model retraining, FRRs could significantly swing over time, attributing to changing typing behaviors and adoption of new typing poses, and affect user experience. An adaptive (model retraining) system is necessary to maintain low FRRs over long-term use. Such an adaptive system, which may be configured to frequently update users' classifiers, should also be responsible for keeping FARs consistent, and provide the same level of security protection during long-term use.
(2) **On-device training.** 12 out of 20 participants expressed privacy concerns with their keystroke data being sent over to a server; 19 participants said they have no concern with the data being stored and processed locally. Hence, all model training and prediction should be performed on phones. Training deep learning models, however, requires large amounts of computational power, memory, and train data, and takes long time to complete – it is impractical to train them directly on smartphones. Training them on a server is also undesirable since most users do not want to share their sensor data. To that end, a more preferred way is to perform "transfer learning" on a device based on a pre-trained deep learning model, and use them as a tool for extracting features.
(3) **Data storage size.** To periodically retrain classifiers, we need to store users' keyboard data on devices. Typically, businesses do not consider authentication solutions that use up more than 100 megabytes of data storage as practical solutions.
(4) **Training time.** User experiences could be hindered severely if periodic retraining is necessary, and if each training takes a long time and slows down phones. The time taken to train the ensemble blender should be within a few minutes.

Our evaluation methodologies are designed to analyze performances against these key requirements and constraints.

3 DESIGN

In this section, we describe the KedyAuth design and discuss our novel feature engineering techniques.

3.1 KedyAuth Overview

Being mindful of users' privacy concerns about their keystroke data being shared (see Section 2.2.4), and following the "on-device training" guideline (see Section 2.3), KedyAuth is designed to train all classifiers on a user's device without sending any sensor data to a server: two binary classifiers are trained with features extracted from the user's sensor data and pre-loaded imposters' (other users') keystroke samples, and the third binary classifier is trained with features computed through the use of a pre-trained common autoencoder (see Figure 3). A data

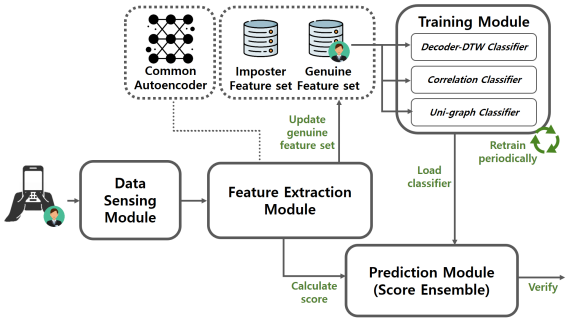


Fig. 3. KedyAuth system overview

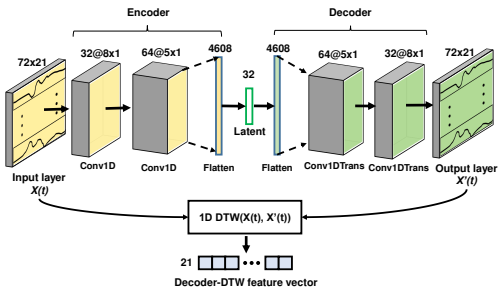


Fig. 4. Autoencoder architecture used to extract Decoder-DTW features.

"sample" refers to a single keystroke sample. Note, KedyAuth ensures the sensitive keystroke data never leaves the device (addressing users' privacy concerns), and complex deep learning models are never trained locally on the device. A KedyAuth-enabled smartphone would be shipped with a pre-trained autoencoder model, and features pre-extracted from the imposter dataset. To initialize KedyAuth before model deployment, we would need to collect some users' keystroke data in a lab setting. Using this data, we would (1) pre-train an autoencoder that is generally good at reproducing given keyboard sensor data, and (2) extract imposter train set features. Considering the "data storage size" guideline (see Section 2.3), we aimed at keeping the feature set size small: the proposed correlation and autoencoder-extracted feature sets consist of just 66 and 21 numeric features, respectively.

We imagine the first phone setup process to be in two phases where a user would be asked to type a few given random sentences in different body poses, including sitting down, walking, and leaning poses. Asking users to type about three sentences (assuming that an average English sentence comprises about 120 letters [26]) for each of the three body poses should be sufficient for the initial blender training. This considers the "consistent performance" guideline (see Section 2.3), ensuring that KedyAuth functions correctly under varying typing and body poses. Around 1,000 (three sentences and three contexts) keystroke samples would be collected manually. In the second phase, we would *automatically* collect more keyboard data while the user is using the phone for a day or two – around 2,000 to 3,000 (about 20 to 30 sentences in total) more samples would be available after a few days of use. Another blender could be trained after a few days using more data.

To ensure that the FRRs (affecting usability) do not deteriorate during long term use (see Section 4.4), the blenders are *retrained frequently* with new data, repeatedly adapting to users' changing typing behaviors and modeling unknown body poses. Note, new samples are collected automatically while the keyboard is being used for daily use (users are never asked to type more sentences for data collection), and the whole retraining process is always *implicit* – never requiring user intervention. Considering the "training time" guideline (see Section 2.3), KedyAuth trains LightGBM classifiers to achieve fast training time and high efficiency: in Section 3.5.6, we demonstrate that the full LightGBM blender can be trained within a few seconds.

At run-time, whenever the default touchscreen keyboard app is activated and the user types on it, KedyAuth will collect the three sensor data, extract features, and use the blender to compute the final authentication score. This score is then compared against a pre-defined threshold, and the phone will be locked if the score is lower than the threshold. This design decision aims to promptly stop unauthorized access, and mitigate the security risks (such as insiders searching for private conversations or sending messages) discussed in Section 2.2.3.

3.2 Feature Overview

We explain the four feature sets that we implemented and evaluated while designing KedyAuth. Two of them are novel feature sets proposed in this paper, and the other two are from prior work such that can also be extracted fully on smartphones. Note, we did not include the HMOG features in the final KedyAuth design based on the accuracy results.

HMOG grasp resistance (HMOG-GR) features [41]. These features measure the resistance of a hand grasp to the force exerted during key tap events using the acclerometer and gyroscope sensors, computing the mean, standard deviation, and difference between sensor values.

Uni-graph features [8]. These features measure the key press duration, contact sizes, phone movement, and touchscreen (X, Y) coordinates during keystrokes. Based on [8], we extract 30 features on the uni-graph of each keystroke from the accelerometer, linear accelerometer, and touch sensors.

Correlation features. One limitation of HMOG-GR features is that each feature contains dynamic information from only one sensor and, as a result, fails to fully capture dynamic keystroke characteristics in higher dimensional space constructed by two or more sensors. For example, whether a peak in sensor A and a peak in sensor B in some time window occur simultaneously or sequentially cannot be discerned by measuring their average and standard deviation over that window separately. We address this problem with a method that extracts information about the joint distribution of pairs of sensors. For all 12 sensors (X, Y, Z, and M directions for accelerometer, linear accelerometer, and gyroscope), we compute Pearson's correlation coefficients between sensor data collected 300 ms before a keystroke's "down" event (the start of a keystroke) and up to 300 ms after a keystroke's "up" event; considering that the sensor frequency is 100 Hz, and a keystroke duration (between "down" and "up" event) is about 10–40 ms, the total sample length is about 60–64 sensor readings; we apply exponential smoothing to this data. We then compute (1) pairwise "intra-correlation" between X, Y, Z, and M within the same (single) sensor; this results in 18 features (e.g., accel-X-Y or gyro-X-Z), and (2) "inter-correlation" across two different sensors, resulting in 48 features (e.g., accel-X-gyro-X or accel-Y-gyro-Z). In total, we experimented with 66 features. We compared the performance of four different algorithms – Pearson's correlation, Spearman, linear least-squares coefficient, and Kendall's tau – with respect to EERs computed on the HMOG dataset for single keystroke, and chose Pearson's correlation as the best performing algorithm. The EERs were 21.6, 22.4, 23.4, and 22.2 percent for the four algorithms, respectively.

Decoder-DTW (DD) features. Another limitation of HMOG-GR and Uni-graph features is that these features are manually engineered extraction of information from raw sensor readings rather than automatically learned representations. To allow automatic learning of most predictive features from data, we used a combination of an autoencoder and dynamic time warping (DTW) [5]. The raw input sensor data consist of about 720 ms of sensor readings for three sensors (accelerometer, linear accelerometer, and gyroscope) in three directions and magnitude (X, Y, Z, and M) and their orientation in the phone's coordinate system (three angles between the sensor planes), resulting in a vector of dimension (72 readings) $\times$ (3 sensors) $\times$ (7 directions) for each keystroke: each input consists of 21 (3×7) separate time series data. As described in greater detail in the next section, we pre-train an autoencoder to transform the raw input vector into an output vector of the same dimensions. Then we compute one-dimension DTW (1D DTW) between input time series (sensor) data and autoencoder-reconstructed output data, representing the magnitude of their differences. We refer to these as "Decoder-DTW" features. The details of how 1D DTW is computed given two time series data are explained in Appendix A.

3.3 Rationale for Feature Selection

We first explored the well-established HMOG-GR features to utilize the accelerometer and gyroscope sensors. Our attempt to optimize HMOG-GR performance through the use of rolling keystrokes (all samples in a given rolling window are used to compute average authentication scores), however, was not so effective: the average

EER measured on the public HMOG dataset mildly fell from 19.5 to 13.7 percent when we increased the rolling window size from one to 70 keystrokes, and used 3,000 samples to train SVM binary classifiers (see Section 3.5.5 for details). We surmise this limited impact is due to the inherent characteristics of the statistical features like average sensor values and difference in sensor values (before and after a tap event): some details about micro phone movements and orientation changes may be ignored while computing them – reducing the value of new information we intended to capture by adding more keystroke data. We excluded HMOG-GR from the final feature set selection.

To overcome this limitation, we studied autoencoder-based anomaly detection approaches [9] that use the raw (accelerometer and gyroscope) sensor values as the input data: a user-specific autoencoder is trained to efficiently compress the original sensor data through the encoder, and reconstruct the data through the decoder that closely resembles the original input. Anomalies can be detected based on the distance between the input data and the reconstructed output data: these distance measurements would be larger for imposters as the autoencoder is less competent in reconstructing unknown keystrokes, and vice versa. Autoencoder-based anomaly detection techniques have been studied and validated extensively in a variety of problem domains, including representation learning [47] and network intrusion detection [12, 29], demonstrating their generalization capabilities and effectiveness in detecting outliers or unwanted events.

Considering the "frequent retraining" needs (see Section 2.1) and "on-device training" guidelines (see Section 2.3), however, training heavy user-specific autoencoders on devices or sending private sensor data to a server and training (or frequently retraining) models on the server should be avoided. To investigate a workaround, we experimented with a widely adopted transfer learning technique [31] that involves (1) training a "common autoencoder" with the keystroke data collected from about 23 users, (2) pre-loading the autoencoder on the device, and using its encoder to compress the input sensor data into 32 dimensionality-reduced (latent space) features for a given user, and (3) training a lightweight user-specific binary classifier based on those 32 features. Even with this deep learning-based feature extraction approach, however, we encountered similar limitations in optimization efforts: increasing the rolling window size from one to 70 keystrokes led to just a small drop in the average EERs (from 21.4 to 17.6 percent).

To facilitate the use of both the encoder and decoder (i.e., the full autoencoder), we adopted the anomaly detection approach described above [9], and *substituted* user-specific models (our design goal is to avoid training them) with the "common autoencoder" trained using keystrokes collected from 23 users. Following the original methods [5, 9], we used the common autoencoder to reconstruct the input sensor data for a given user, and measure the distance (1D DTW) between the input and reconstructed output. A lightweight user-specific binary classifier is trained based on these distance measurements, along with a pre-loaded imposter train set comprising the same features extracted from 20 other users. The key notion is preserved: the target user's distance measurements, referred to as the DD features (see Section 3.2), represent *how well the common autoencoder reconstructs the user's input data.* The binary classifier is trained to distinguish these reference measurements (uniquely representing the target user) from other measurements that represent different competencies in which the common autoencoder reconstructs imposters' keystroke data. As shown in Section 3.5.5, increasing the rolling window size from one to 70 keystrokes led to a significant elevation in authentication (prediction probability) scores for genuine users – this optimization effort, in turn, led to an impressive reduction (from 21.3 to 7.7 percent) in the average EERs. The "data storage size" guidelines (see Section 2.3) were also considered while designing the DD features: a direct use of the autoencoder would involve storing raw input sensor data (72 readings × 3 sensors × 7 directions per keystroke) for retraining purposes; our DD features compress such complex time series data into just 21 numeric features, and save disk space.

Additionally, we explored the use of uni-graph features [8] to utilize touch sensors, and extract information about the touchscreen in-contact positions, drag distances, key tap and hold duration, and in-contact area sizes.

Prior efforts, including HMOG-GR and uni-graph extraction methods, have primarily focused on extracting independent one-dimensional features from a single sensor – overlooking the possibilities of modeling more complex keystroke dynamics and characteristics through the combined use of two or more sensors. To bridge this gap, our correlation features have been designed to model more complex, multi-dimensional relations (correlations) between two different sensors. For instance, HMOG-GR measures the resistance of a hand grasp (to the forces exerted by key taps) based on micro movements measured through the accelerometer, and, separately, phone orientation changes measured through the gyroscope [41]. Our intuition is that the "correlation strength" between phone movements and orientation changes (i.e., between the two sensor values) would be affected by factors such as holding postures, hand-grip strengths, tapping strengths, and so on, and would be uniquely different for each individual: as such, we intended to capture these unique correlation strengths by computing pairwise inter-correlations between the gyroscope and accelerometer sensor values during a tap event. Unlike the one-dimensional HMOG-GR features, these correlation features demonstrated a significant drop in the average EERs (from 20.9 to 7.6 percent) when we increased the rolling window size from one to 70 keystrokes.

3.4 Training Models

3.4.1 Pre-training. The pre-training phase involves collecting keyboard use data from some number of users with well informed (future data use) consents and remuneration. Since the requirement is to keep this number minimal, we roughly estimate that collecting data from about 50 people (with balanced demographics) with around one-hour data collection session per person would be a reasonable overhead. Assuming that data for about 50 people are available, we split it into two halves, use one half as the other user set (to be deployed on users' devices), and the other half to pre-train autoencoder and also for finding optimal hyperparameters. The autoencoder is trained with the input vector shape described above. Its encoder consists of two one-dimensional fully convolutional neural network (FCN) blocks [19], each with 32 and 64 filters and kernel size 8 and 5, respectively (see Figure 4). We applied batch normalization and ReLU activation function after each FCN block. We also applied 0.1 dropout rate. The latent space size for the flatten layer is 32. The decoder is trained to regenerate output values that closely resemble the given input vector. We used mean squared error (MSE) [22] as the autoencoder loss function.

The global hyperparameters (to be used for training the user-specific binary classifiers) are optimized using the other user set as the imposter train set and the remaining set (the same set used to train the autoencoder) as the genuine user set. We used the "hyperopt" package [4] in Python to find optimal hyperparameters.

3.4.2 On-device Training. Then there is the local, on-device training phase. Once about 1,000 user keystroke samples are collected locally, all the feature sets are extracted as described in Section 3.2. The number of genuine user samples and the number of imposter samples are equally balanced. Using this balanced train set and the global hyperparameters, we train three binary classifiers on the device. The final ensemble blender that uses all classifiers together is not trained (this would require additional data) – rather, we take a simple average across all three classification scores to compute the final authentication score, and use 0.5 as the default threshold value to make authentication decisions.

3.4.3 Retraining. To adapt to changing typing behaviors (e.g., typing speed) over time, the full blender is periodically retrained with the latest data available (the imposter train set is fixed and reused). The entire retraining process is always *implicit*: new keystrokes are automatically sampled from the latest typing data (available from daily use), and the classifiers can be retrained as a background process without any user intervention when certain number of new samples have been accumulated, or when the phone screen is turned off and it is not being used.

3.5 Performance Optimization

In this section, we evaluate the performance of the four feature sets (HMOG-GR, Uni-graph, Correlation, and DD) based on the HMOG dataset [41], and find the optimal ensemble blender configuration with respect to EERs. We also experiment with different classification algorithms and evaluate system overheads to find an algorithm that best satisfies the deployment constraints (see Section 2.3).

3.5.1 Software and Hardware Used. All EER evaluations were conducted using Python on Ubuntu 18.04 (64-bit). The scikit-learn library [33] was used to train Support Vector Machine (SVM) classifiers, and the Light Gradient Boosting Machine (LightGBM) package [24] was used to train LightGBM classifiers; hyperopt [4] was used to optimize hyperparameters. We used Keras [14] for autoencoder implementation and dtw-python package [21] to compute 1D DTW. To measure model training time, size, and prediction time, we constructed a prototype implementation in Java and C that fully runs on Samsung Galaxy S20 Plus, comprising the feature extraction module, classifier training module, and prediction module described in Figure 3. For this assessment, we used the processed HMOG dataset as the initial input data, feeding them into the feature extraction module to start the evaluation process.

3.5.2 HMOG Dataset. The HMOG dataset [41] is a popularly used public touchscreen dataset collected through a session-based lab study. A total of 100 participants were recruited in the United States. Through 8 different data collection sessions, the participants were asked to perform a given set of freely typing tasks while sitting or walking. All participants typed in English. On average, there are about 6,310 ($\sigma = 1,380$) keystroke samples from each participant, where about 50 percent of the samples are associated with the "sitting" context; the rest are associated with "walking" context. To split the user set into (1) other user (imposter) set, (2) autoencoder training set, and (3) genuine user (evaluation) set, we first selected 57 genuine users who had enough samples (in both "sitting" and "walking" contexts) for our specific evaluation settings; we then selected 20 users with the least number of samples as the imposter set, and the remaining 23 as the autoencoder set.

3.5.3 Setup. To measure EER, we train four binary classifiers for each user in the evaluation set using the optimal hyperparameters. To create a train set, we select initial 1,000 (or more based on the evaluation setting) keystroke samples from a target/genuine user with balanced context – half from the "sitting" context and the other half from the "walking" context. Train samples are selected from the initial few sessions to preserve event sequence (temporal) information. Similarly, we select the same number of context-balanced samples from the imposter set – since there are 20 imposters, we proportionally select samples from each imposter, with a random starting index, to fit the overall train set size. This imposter train set is fixed, and the same set is used to train classifiers for all users in the evaluation set.

To create test sets, for each target/genuine user, we selected 1,000 test samples from the end of the dataset consisting of both "walking" and "sitting" contexts proportionally. The idea was to test the model performance with the same test set regardless of the size of train samples, and also with the most challenging test set – samples that are furthest away from the train samples. To select the other user (imposter) test set, for each of the other 56 users, we used the same 1,000 test samples selected as their own genuine test set. As a result, classifiers were tested using 1,000 genuine user test samples and 56,000 imposter test samples.

3.5.4 Authentication Accuracy. Based on the settings described in the previous section, we evaluated average EERs across all 57 users. We experimented with SVM (RBF kernel) and LightGBM as two suitable classification algorithms: SVM is widely used to solve binary classification problems, and LightGBM is a decision-tree based algorithm designed to optimize model training time and size. We used a rolling window size of 30 and 70 keystrokes for this evaluation, and experimented with different train set sizes. We measured average EERs for each individual feature set, and all possible blending (ensemble) combinations to find the best-performing blender.

Table 3. Average EERs and standard deviations (SD) computed with the HMOG dataset for single classifiers (both for SVM and LightGBM) as well as top performing blenders. We show three top performing blenders as well as a baseline blender that uses just the state-of-the-art features (GR, Uni) while increasing number of train samples and rolling keystrokes; e.g., "1K/30" implies that a model was trained using 1,000 samples, and 30 rolling keystrokes were used to compute final authentication scores.

Feature	Average EERs (SD)							
	# training data / # rolling keystrokes (SVM)				# training data / # rolling keystrokes (LightGBM)			
	1K / 30	1K / 70	3K / 30	3K / 70	1K / 30	1K / 70	3K / 30	3K / 70
GR	18.51 (13.92)	17.70 (14.15)	14.63 (12.68)	13.69 (12.77)	22.19 (15.41)	21.43 (15.84)	18.93 (15.49)	18.05 (15.62)
Uni	16.09 (12.97)	15.01 (13.52)	13.80 (10.46)	12.57 (10.53)	18.53 (15.23)	17.38 (15.50)	13.67 (12.31)	12.63 (12.39)
Corr	11.70 (13.14)	10.42 (13.69)	8.88 (10.30)	7.57 (10.41)	15.03 (12.88)	13.47 (13.16)	11.94 (09.97)	10.67 (10.41)
DD	14.76 (13.11)	13.22 (13.21)	9.28 (08.41)	7.65 (08.33)	15.84 (13.84)	14.40 (14.40)	9.45 (09.04)	7.95 (09.09)
Latent	21.90 (15.67)	21.12 (15.85)	18.30 (13.84)	17.60 (13.98)	22.69 (14.83)	21.85 (14.94)	18.90 (13.83)	18.13 (14.05)
GR, Uni	14.98 (12.75)	14.15 (12.96)	12.25 (10.80)	11.25 (10.74)	19.33 (15.37)	18.50 (15.52)	14.90 (13.39)	14.10 (13.45)
Uni, Corr	9.78 (12.63)	8.92 (12.95)	7.16 (09.42)	6.12 (09.61)	12.54 (12.64)	11.49 (12.76)	9.18 (10.27)	8.24 (10.43)
Corr, DD	9.74 (12.65)	8.42 (12.52)	6.03 (08.97)	5.01 (08.84)	11.46 (12.60)	10.17 (12.71)	7.42 (09.19)	6.40 (09.22)
Uni,Corr,DD	9.25 (11.80)	8.25 (11.72)	5.68 (08.39)	4.91 (08.69)	10.61 (11.75)	9.47 (11.93)	7.27 (09.11)	6.57 (09.30)

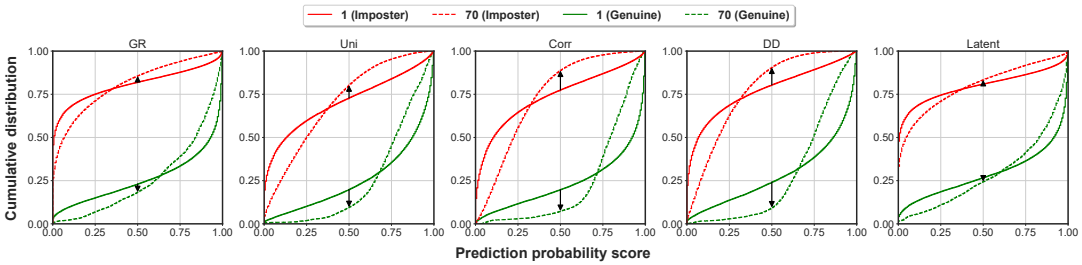


Fig. 5. CDFs computed on the prediction probability scores with respect to 1 and 70 rolling keystrokes. We show both genuine (green) and imposter (red) test set CDFs for all five features sets.

Our results are summarized in Table 3. As for the individual models, `SVM(Corr)` performed the best, achieving 7.6 percent with 3,000 train samples. In comparison, `SVM(DD)` achieved 7.7 percent; `LightGBM(DD)` was the best performing LightGBM model, achieving 8 percent. Both `SVM(GR)` and `SVM(Uni)` showed higher EERs at 13.7 and 12.6 percent, respectively. The top two SVM blenders are `SVM_blend(Corr,DD)` and `SVM_blend(Uni,Corr,DD)` with average EERs of 5 and 4.9 percent, respectively. Similarly, the top performing LightGBM blenders are `LightGBM_blend(Corr,DD)` and `LightGBM_blend(Uni,Corr,DD)` with slightly worse performance: 6.4 and 6.6 percent. Results in Table 3 suggest that using all three `(Uni,Corr,DD)` features demonstrates more consistent performance under varying classification algorithms, train set sizes, and rolling window sizes. Note, we also present a baseline blender that only uses existing feature sets: HMOG-GR and Uni-graph features – this blender, `SVM_blend(GR,Uni)`, achieved a much higher average EER at 11.3 percent. These comparisons illustrate the effectiveness of the two feature sets (Correlation and Decoder-DTW) we propose in the paper. We performed Wilcoxon signed-ranked tests with Bonferroni correction between the EER distributions – `SVM_blend(Corr,DD)` vs. `SVM(GR)`, `SVM(Uni)`, `SVM_blend(GR,Uni)` – to show statistically significant superiority in all comparisons ($p < 0.0001$).

3.5.5 Keystroke Rolling Effects. Figure 5 shows cumulative distribution function (CDF) of the genuine (green) and imposter (red) test scores for the 5 feature sets while varying the "number of rolling keystrokes" (used to compute average probability scores) between 1 and 70. The black vertical arrows indicate the differences in the

Table 4. Feature and model sizes, training times, and prediction times measured after deploying the classifiers on Samsung Galaxy S20 Plus. We show the average and SD values.

Feature	Uni		Corr		DD		All three	
Algorithm	**SVM**	**LGBM**	**SVM**	**LGBM**	**SVM**	**LGBM**	**SVM**	**LGBM**
Feat. + model size (MB)	3.47 (0.03)	1.93 (0.06)	9.23 (0.02)	3.78 (0.00)	6.07 (0.00)	6.20 (0.00)	18.77 (0.04)	11.90 (0.06)
Total train time (s)	10.81 (0.12)	1.11 (0.21)	39.99 (0.34)	2.61 (0.28)	6.15 (0.22)	0.92 (0.23)	56.94 (0.40)	4.63 (0.41)
Prediction time (ms)	0.15 (0.07)	0.13 (0.05)	0.53 (0.35)	0.46 (0.27)	18.14 (4.80)	17.31 (4.37)	18.82 (4.90)	17.91 (4.46)

Best blender	**LGBM_blend(Uni,Corr,DD)**		
Train set size	**3K**	**5K**	**10K**
Feat. + model size (MB)	11.90 (0.06)	16.00 (0.03)	26.22 (0.04)
Total train time (s)	4.63 (0.41)	5.60 (0.46)	6.84 (0.73)
Prediction time (ms)	17.91 (4.46)	19.07 (8.61)	16.84 (4.29)

two plots at 0.5 prediction probability score: as for Uni-graph, Correlation, and Decoder-DTW feature sets, we observed a significant drop in the CDFs for the genuine test scores, indicating that we can effectively boost the genuine scores by increasing the number of keystrokes. As for the imposter test scores, we observed a significant increase in the CDFs at 0.5, indicating that by increasing the number of keystrokes to 70, we can effectively lower the overall imposter scores for those three feature sets – these trends explain the significant improvements in accuracy. Such effects were much smaller in HMOG-GR and Latent Space, showing smaller decrease or increase in the CDFs at 0.5. As a result, the overall accuracy improvements were less significant.

3.5.6 Overheads. We implemented all classifiers and the most reliable blenders, `SVM_blend(Uni,Corr,DD)` and `LightGBM_blend(Uni,Corr,DD)`, in C and Java to deploy them on Samsung Galaxy S20 Plus, and measured model training time, prediction time, and model sizes. We used 3,000 samples for training, and one keystroke for prediction. We measured the average training time across all 57 users in the evaluation set, and measured average prediction times by selecting 200 test samples from each of the 57 users. A summary of the results are presented in Table 4. The full LightGBM blender, compared to SVM, was more efficient with respect to the model and feature size (11.9 vs. 18.8 megabytes), train time (4.6 vs. 56.9 seconds), and prediction time (17.9 vs. 18.8 milliseconds). Being mindful of the data storage size and training time guidelines (see Section 2.3), and the need to frequently retrain models (see Section 2.1), we recommend using LightGBM for deployment – the trade-off is about 1.7 difference in the average EERs (4.9 vs. 6.6 percent). Even with a much larger train set consisting of 10,000 samples, LightGBM takes about 6.8 seconds to train the full blender.

4 EVALUATION 1: REAL-WORLD ACCURACY

This section explains the real-world keystroke dataset we collected over a two-month period to evaluate the long-term performance. Based on this dataset, we evaluate (1) how the best blender performance changes over time, and (2) how retraining strategies can be applied to stabilize long-term error rates.

4.1 Real-world Data Collection

To evaluate how the model accuracy changes over time, we collected real-world touchscreen use data by recruiting 40 participants from Ukraine – handing out Samsung Galaxy S9 devices with a customized keyboard app and logging app installed, and asking them to use these devices as their only (primary) device for two months. We also asked the participants to only use the default (our custom) keyboard. We had to modify the Android Window Service Manager, and install this modified firmware on those devices – this modification was necessary to collect touch events across all applications. We freely allowed all application updates. All participants inserted their own SIM cards into the phones we handed out, and used them throughout the study. We checked daily lock/unlock logs to ensure that the participants were actively using the custom keyboard. Note that this study was designed to collect daily keyboard use data solely; authentication functions were not implemented.

We recruited participants through street advertisements and social networks to ensure a wide range of occupations, and maintain a balanced female/male ratio. 19 (out of 40) participants were male, and the average age

was 31.9 ($\sigma = 8.3$). All of the participants were white. 32 participants said they unlock their phones a few times per hour. Occupations varied a lot, including sales, computer, legal, education, and so on. We did not implement any feature related to letters or words being typed and believe that education levels would have had a small impact on accuracy results. The full demographic details are in Appendix B.

Our data collection app collected gyroscope, accelerometer, linear accelerometer, and touch sensor data whenever our custom QWERTY keyboard was active and being used. With respect to keyboard events, we collected timestamp and event type (keyboard open/close/tap, and switch language). All data were first stored locally on the phones, and uploaded automatically to a data server on a daily basis. We did not restrict language options, allowing participants to use multiple languages of their preferences. Most of the participants used Ukrainian and Russian, and some also used English interchangeably. People in Ukraine and Russia commonly use QWERTY keyboards to type Ukrainian, Russian (these two are almost identical), and English. All of the participants were familiar with the QWERTY layout. We did not encourage the participants to type more. Each participant was compensated with the equivalent of USD 200 in local currency (7,300 Hryvnia) for completing the study. On average, we collected 65,000 ($\sigma = 82,000$) valid keystroke samples from each participant. About 40 percent of the samples are associated with "still" contexts, and the rest are associated with "moving" contexts. We used the "Android activity recognition API" [35] to create contexts labels – these labels are used to create a train set with mixed and balanced contexts. We filtered out data representing static phone placements like "on desk" contexts because these positions result in near-zero movement sensor readings with a high signal-to-noise ratio. We tested this data collection and log monitoring protocol on seven pilot study participants for several days, fixing all software bugs and protocol flaws before recruiting the actual study participants.

Again, we sorted the participants based on the number of samples available and excluded 5 participants who had the smallest number of keystroke (between 1,256 and 4,444) samples. We then selected 20 participants to be included in the evaluation set: these participants had enough samples (in both "still" and "moving" contexts) for our 10 temporal (sequential) session evaluation. We experimented with 10 random genuine set permutations, randomly selecting 10 genuine users and 10 imposter train users each time. The remaining 15 participants were used as a fixed autoencoder train set.

To demonstrate the validity of our dataset, we compare some basic statistics of our dataset against other real-world datasets. For instance, the average keystrokes recorded per day presented by [7] was 1,529 ($N = 30$) compared to 1,076 ($N = 40$) measured in our dataset. Further, the average phone unlocks per day presented by [23] was 39.9 ($N = 134$), and it was 50 in our dataset. In both papers, the participants were using their own phones.

4.2 Ethical Considerations

While collecting the real-world dataset, we explicitly informed the participants that the purpose of the data collection was to develop and evaluate an implicit authentication solution using keystroke dynamics. The ethical perspective of our research was validated through an institutional review board (IRB) at a university. Further, we inquired and checked with a user privacy protection office, and received confirmation that there are no legal and privacy issues in utilizing the collected data for research so long as we do not present user identifiable information.

We asked the participants to perform a factory reset before returning the devices – we also helped them perform a factory reset during their visit to ensure that all devices were properly reset/wiped, and that no sensitive data like passwords remained. Our phones automatically uploaded the collected sensor data to our external server on a daily basis through HTTPS, and deleted them as soon as daily uploads were complete. Our external server was password-protected, and only those involved in this paper were able to log in, and access the data. The researchers then used secure connections to transfer all the data from the external server to an internal server

(again, on a daily basis), which was protected inside a private network; our organization-wide security policy prohibits all external data transfer. We immediately deleted all the data from the external server. With respect to the keyboard events, we completely deleted all key-codes during data pre-processing. Further, we did not store any personally identifiable information.

4.3 Setup and Retraining Strategies

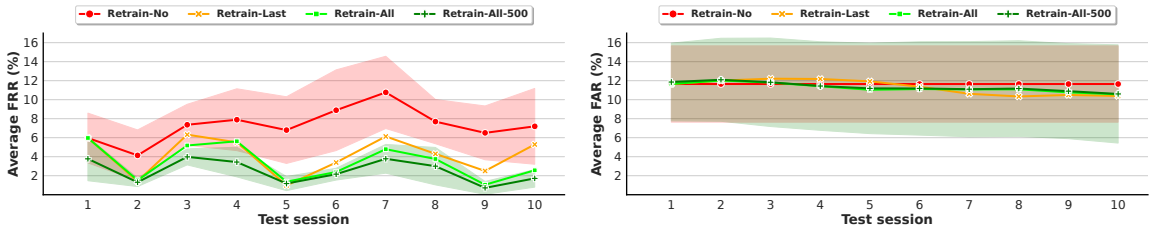
We split the data into three separate sets (see Section 4.1): 20 participants who have enough samples for our 10 temporal test sessions were included in the evaluation set, and the remaining 15 participant data were used as the fixed autoencoder train set. To reduce genuine user sampling bias, we experimented with 10 random genuine set permutations: to create a permutation set, we randomly selected 10 genuine users and 10 train set imposters (from the 20 participant evaluation set). We ensured that all 20 participants have been included in the genuine set at least once. We selected the most efficient and reliable blender, `LightGBM_blend(Uni,Corr,DD)`, and measured FRRs and FARs over 10 temporal "test sessions," each containing 1,000 test samples.

To set up a temporal testing environment with the two-month data, we fixed the number of samples for training a blender for each of the 10 genuine users to 5,000 (2,500 from each of the two "still" and "moving" contexts). We decided to experiment with a larger train set size (compared to 3,000 used in the first evaluation) as the genuine users had sufficiently large number of samples available. As for the other user (imposter) train set, we selected 5,000 samples from the 10 other users based on a random sampling (start) index. For each genuine user, the next 1,000 samples (following the 5,000 used for training) were used as the first test session set – again 500 were selected from each of the two contexts. Then the following 1,000 samples were used as the second session test set; we repeated this process until all 10 temporal test sessions were evaluated sequentially. Our evaluation set split method ensured that genuine users in all 10 permutations had enough samples to create at least 10 sequential test sessions – we fixed the total number of test sessions to 10, considering the number of samples available for the participant with the least number of samples. To select the imposter test set, we selected 1,000 samples (mixed contexts) from each of the remaining 9 users, creating a large, fixed imposter test set. We used the first 10 testing sessions of genuine test set and the fixed imposter test set to compute average FRRs and FARs. Only the genuine test set samples change across the 10 testing sessions. We repeated this evaluation for 10 random permutations, and computed average FARs and FRRs across all permutations for each of the 10 test sessions.

We experiment with three retraining strategies to investigate how training data size and training frequency affect model accuracy. The first retraining strategy (Retrain-Last) involves using the *last* collected samples. For each of the 10 genuine users, we used the last 5,000 samples (2,500 selected from each context) available to retrain models for every testing session. The first testing session is evaluated as described above. In the second testing session, a new model is trained with the 1,000 samples from the first testing session and 4,000 samples from the original 5,000-sample train set (the oldest 1,000 samples are excluded), and evaluated using the second test set. We repeat this process for all testing sessions.

The second retraining strategy (Retrain-All) involves using the entire set available to retrain classifiers. The experimental setup is the same as the first retraining strategy except that we use all the samples collected until the start of the next testing session to train classifiers. We cannot increase the train set size endlessly due to the "data storage size" and "training time" requirements (see Section 2.3). Hence, we stop increasing the train set size when it reaches 10,000, and use the last 10,000 samples from that point onward. Another difference in the setup is that we also increase the imposter train set size to match the genuine user train set size.

As for the third retraining strategy (Retrain-All-500), we apply the second strategy but increase the retraining frequency: for each 1000-sample test session, we train a new model after 500 samples; i.e., each test session is evaluated with two different models (switching to a new model after 500 samples).



(a) Average FRRs (affects usability)

(b) Average FARs (affects security)

Fig. 6. Average FRRs and FARs measured using 0.5 threshold, across the 10 test sessions. We show SD bands across 10 permutation rounds for "Retrain-All-500" and "Retrain-No."

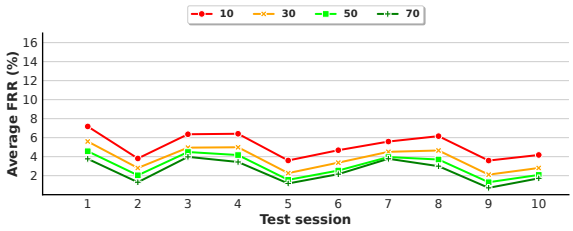


Fig. 7. Average FRRs measured across the 10 testing sessions with varying number of keystrokes, 10, 30, 50, and 70.

4.4 FRR and FAR with Retraining

We measured average FRRs and FARs for each of the 10 testing sessions (i.e., across 10 genuine users), and computed average across 10 permutations – experimenting with the three retraining conditions (Retrain-Last, Retrain-All and Retrain-All-500) and a baseline (Retrain-No) condition. We used a sliding window of 70 consecutive keystrokes to compute final authentication scores, and used 0.5 threshold to measure FRRs and FARs. Unlike prior work, we do not search for optimal threshold values tailored to the studied genuine user set. Finding optimal thresholds adds more complexity to deployment and validation: it is unclear what samples and strategy should be employed to search for them, and how sensitive they will be to unknown typing behaviors and attack sets. We take a more practical approach instead, and use an *unbiased*, preset threshold of 0.5 for this particular analysis. Note, other preset values like 0.4 or 0.6 may also be selected to facilitate a more user-friendly mode or a more secure mode based on given system deployment requirements – this trade-off is discussed in Sections 6.3 and 4.6.

As shown in Figure 6, without any retraining (Retrain-No), the average FRR is above 5 percent for 9 (out of 10) testing sessions, and reaches 10 percent at the 7th test session, demonstrating that users' frustration could grow over time; increasing standard deviations (red bands) indicate that some users may face significantly more false rejections. The average FRR for Retrain-No was 7.3 percent (σ = 3.5). Our real-world FRR analyses in Section 2.1 show that this is likely due to changing typing behaviors: the blenders trained once during initial setup would gradually become less accurate and relevant as users' typing behaviors evolve over time. In Section 4.7, we also explain the effects of unknown typing contexts like body poses on accuracy: the blenders will perform poorly on new contexts and poses that have not been captured as part of the initial setup process. Frequent retraining (Retrain-All-500) was most effective – lowering average FRRs in all testing sessions and long term use. Its average FRR was 2.5 percent (σ = 1.7) with much smaller standard deviation (green) bands. With the Retrain-Last and Retrain-All strategy, the average FRRs were 4.2 percent (σ = 2.6) and 3.4 percent (σ = 2.4).

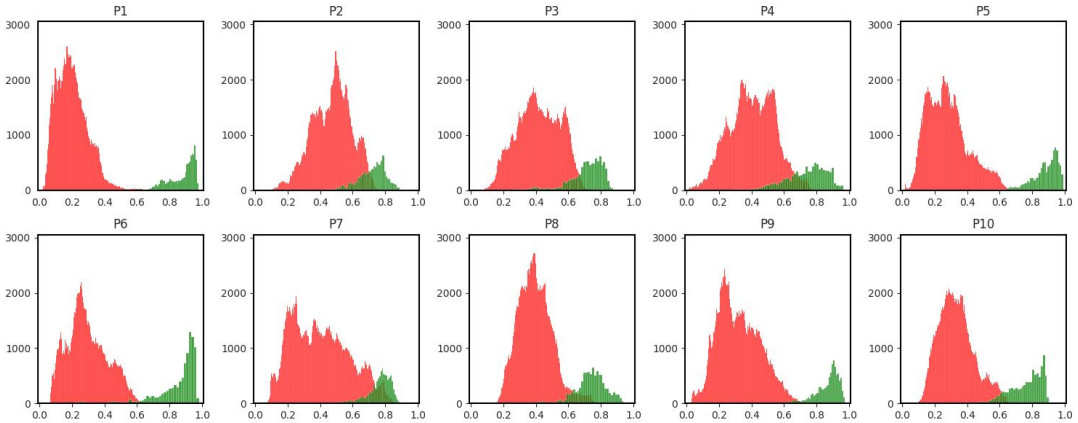


Fig. 8. FCS for the 10 genuine users in a permutation round that showed the highest average HTER.

These results demonstrate the importance of retraining: frequent retraining ensures that the blenders promptly adapt to users' evolving typing behaviors and new contexts and body poses, and keep FRRs low during long term use. The average FARs across all test sessions and permutations for Retrain-No, Retrain-Last, Retrain-All and Retrain-All-500 were 11.7, 11.3, 11.3, and 11.3 percent (with 0.5 threshold). Figure 6b shows consistent FARs across the 10 test sessions for all retraining strategies – satisfying the "consistent performance" guideline (see Section 2.3), and ensuring that security is not compromised through frequent retraining.

Figure 7 shows how average FRRs are affected with varying number of rolling keystrokes (we tried 10, 30, 50, and 70) – only the best performing Retrain-All-500 retraining strategy was used for this evaluation. Results for threshold 0.5 are the same as those shown in Figure 6. Obviously, the smaller the number rolling keystrokes needed for initial scoring the faster the attack detection. With 10, 30 and 50 rolling keystrokes, the average FRRs were 5.2 percent ($\sigma = 2.0$), 3.8 percent ($\sigma = 1.8$) and 3.0 percent ($\sigma = 1.8$) respectively. These are moderate increases from the 2.5 percent ($\sigma = 1.7$) we observed when 70 rolling keystrokes were used. As for FARs, starting from 11.3 percent (with 70 keystrokes), the average FARs increased by 3.7, 1.8 and 0.8 percent, respectively.

4.5 Prediction Probability Score Distributions

We selected a permutation set that showed the highest average half total error rate (HTER) across all 10 test sessions, and computed "frequency count of scores" (FCS) [43] for each of the 10 genuine users as shown in Figure 8. HTER is computed as $(FAR + FRR)/2$, and it was 9.9 percent in that highest case. We plotted probability scores from all test sessions that have been computed using the Retrain-All-500 strategy and 70 rolling keystrokes. Overall, these graphs show that the binary classifiers are trained in a desirable manner: in general, the genuine test samples show left-skewed distributions above 0.5, and the imposter test samples show right-skewed distributions below 0.5. With P2, P3, P4, and P7, however, noticeable portions of imposter test samples have scores higher than 0.5, contributing to relatively high FARs when 0.5 is used as the threshold. These results indicate that a preset threshold higher than 0.5 may have to be used (to maximize protection) if businesses are operating in more strict, security-critical environments. We discuss this tradeoff in the next section.

4.6 FAR and FRR Tradeoff

Figure 9 presents receiver operating characteristic (ROC) curves for the 10 genuine users that have been selected from the random permutation set that showed the highest average HTER. These ROC curves demonstrate the

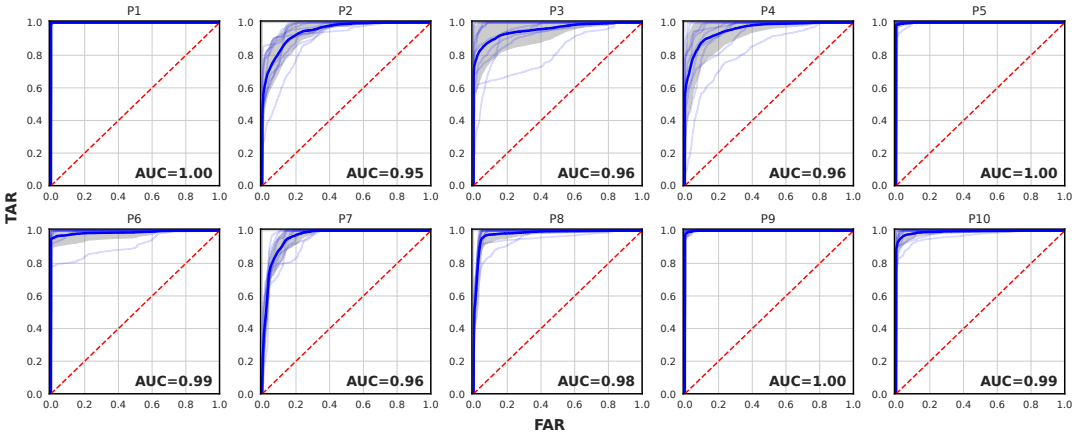


Fig. 9. Average ROC curves for the 10 genuine users in the permutation set that showed the highest HTER.

tradeoff between FAR and FRR across all 10 test sessions. We used Retrain-All-500 with 70 rolling keystrokes for this analysis. The dark blue lines represent average ROC curves, and the lighter lines represent each individual test session. The objective of this analysis is not to suggest a preset threshold that would guarantee certain levels of FRRs or FARs. Rather, these ROC curves illustrate how preset values may be configured to satisfy different business requirements: e.g., presetting 0.4 threshold would facilitate user-friendly mode with about 0.8 percent FRR and 73 percent attack detection rate.

4.7 Unknown Context and Untraining Effects

In this section, we show the consequences of using context-biased training samples, and why it is necessary to use context-balanced train sets. Based on the 10 genuine users selected for each permutation round, we performed cross-training evaluations with "still" and "moving" contexts. For each user, we used the first 5,000 still samples to train a blender, and tested its performance against 2,500 moving samples (selected after the first 5,000 moving samples, which were left out for cross-training), and vice versa. To show the effects of mixing contexts, we then trained a blender with 2,500 samples from each context, and tested with all 2,500 test samples from both contexts. Table 5 shows the average error rates across all genuine users from 10 permutations: context-biased training showed 15.9 percent FRR when a blender was trained with still samples and tested with moving samples, and 16.9 percent FRR in the opposite test setting. Such high error rates demonstrate the risk of (1) potential "untraining effects" where classifiers are retrained with context-biased samples, and incur high error rates in other usage contexts, and (2) "unknown context effects" in which usage contexts that have not been trained may incur high error rates.

By mixing the two contexts, we can effectively mitigate this effect, lowering FRRs to 3.4 percent and 6.7 percent, respectively. Note, we used context-balanced train sets for all retraining sessions in Section 4.4. These results emphasize the importance of covering a variety of typing postures during initial setup, continuously collecting context-labeled samples during use, and using all of those samples together in a context-balanced manner during retraining.

Table 5. Cross-training evaluations for "still" and "moving" contexts. We show the average FRRs and FARs with SD.

Train Context	Test Context	Avg. FRR (SD)	Avg. FAR (SD)
Still	Moving	15.93 (8.03)	10.15 (3.92)
Still, Moving	Moving	3.37 (2.97)	11.76 (5.63)
Moving	Still	16.89 (6.08)	10.93 (4.71)
Still, Moving	Still	6.65 (4.29)	10.92 (4.15)

4.8 Language Dependency Analysis

From our real-world dataset, two users in the genuine user list had a sufficient number of English keystrokes to perform cross-training on two different languages. Since Russian and Ukrainian are almost the same, we treat them as one language and perform cross-training and testing using English keystroke data as another language. First, we trained a blender for each user using just 1,000 Russian/Ukrainian samples, and tested its performance on 1,000 Russian/Ukrainian test samples and 1,000 English test samples separately. By using 70 consecutive keystrokes, for one of the two users, this blender achieved an EER of 4.9 percent on the Russian/Ukrainian test set, and 7.9 percent on the English test set; for the other user, the results were 4.2 percent and 3.4 percent EER, respectively. Second, we repeated that experiment but trained a blender using 1,000 English samples. For one user, the EERs computed on the Russian/Ukrainian test set and English test set were 7.7 percent and 8.0 percent, respectively. For the other user, the EERs were 0.2 percent and 1.6 percent, respectively. Although we only had two users to experiment with, these preliminary results indicate that KedyAuth might be robust in multi-language environments provided that the keyboard layout is the same.

4.9 Setup Time

To gauge how long it would take for users to complete the initial setup process (see Section 3.1), from the 20 evaluation set users, we selected all typing sessions that contain at least 100 continuous keystrokes – sessions are artificially divided by identifying 3 second (or longer) time gap between two keystrokes – and analyzed the time it took for each user to type 100 keystrokes continuously. As mentioned in Section 3.1, we imagine that users would type about 1,000 keystrokes during the initial setup, and the rest of the train samples would be collected implicitly while users use their phones. For the slowest typing user, it took 84 seconds on average to type 100 keystrokes (roughly 840 seconds for 1,000 strokes), and for the fastest typing user, it took 44 seconds on average (440 seconds for 1,000 keystrokes). Note, these are rough estimates, and actual setup time measured through a real setup UI may be different – for instance, a UI may be designed to display random sentences that users can simply read and type; such setup UIs would require less time. Alternatively, if users are asked to come up with their own sentences to type, it may take much longer than the estimated times.

5 EVALUATION 2: RETRAINING OVERHEADS

We evaluated the memory and battery usage of the retraining-enabled KedyAuth system through a lab study involving 9 participants. We used the prototype implementation described in Section 3.5.1 and assessed the performance of the Retrain-All-500 strategy, which trains a new model when 500 new samples are available.

5.1 Methodology

To conduct this study, we extended the prototype implementation described in Section 3.5.1 – adding a run-time raw data sensing module and a simple user interface that comprises a standard QWERTY keyboard and a blank typing screen. The same device, Samsung Galaxy S20 Plus, was used again. We prepared three different KedyAuth configurations for a comparative analysis: (1) KedyAuth is disabled, (2) KedyAuth is enabled, and performs

data sensing, extracts features, and predicts authentication scores at run-time while participants type, and (3) KedyAuth is enabled to perform all of the above functions, and, in addition, retrains the full LightGBM blender at run-time when 500 new samples become available.

The participants were given a long passage containing about 1,500 characters to type, and asked to type the passage as precisely as possible (we did not mandate 100 percent conformance). The average number of keys entered per day over the 20 participants in the evaluation set was 1,214.8 (SD=1,109.2), and the median was 713. We roughly chose 1,500 characters to represent a length that would sufficiently cover most participants' daily use, and also measure the effects of performing at least three rounds of model retraining.

The participants completed two separate sessions. In the first session, they were asked to type the full passage twice: once based on the first (KedyAuth disabled) configuration, and another round based on the second (KedyAuth enabled but without retraining) configuration. We randomized the order in which they performed those two typing tasks to minimize priming effects. After the first session the participants completed a short survey consisting of the following question: *"Did you experience more keyboard lags in one typing task over the other?"* We explained keyboard lags refer to any keyboard loading or rendering delay, freeze, slow down, or characters appearing on the screen a few seconds after entering them on the keyboard. If they said *"Yes,"* we asked more follow-up questions *"In which task did you experience more lags?" "How perceptible was the difference?"* and *"How many times did you experience the lag?"*

After completing the survey, the participants completed the second session, which involved typing the full passage twice based on the first and third (KedyAuth and retraining both enabled) configurations. Again, we randomized the KedyAuth configuration order. For every typing task and KedyAuth configuration, we measured the time taken to complete the task, memory usage rate, and battery usage rate. To measure the battery usage rate, we used the Android dumpsys tool [34], which was configured to measure the battery solely consumed by KedyAuth. We measured the total battery usage rate in percentage upon completion of each typing task. We also used same tool to log the memory usage status every 10 seconds while users completed the typing tasks, and computed the average usage rate for each task. After completing the second session, the participants were provided with a final survey, asking the same question (see above) about their experiences with keyboard lags. The entire study took about 42 minutes (SD=9) to complete, and the participants were compensated with a gift card worth the equivalent of 8 USD in local currency. This study protocol was reviewed and approved by the university's IRB.

5.2 Demographics

We recruited a new group of 9 participants at a large IT company through the use of company mailing lists and online notice boards. Five were male and four were female. The average age was 33 (SD=6.2). The occupation of one participant was related to office and administrative support; the rest had occupations related to computer and mathematics. Full details can be found in Table 6.

5.3 Memory and Battery Usage

To evaluate the system overheads associated with our Retrain-All-500 scheme, we measured battery and memory usage rates for KedyAuth with and without retraining enabled across nine participants. The experimental results are presented in Figure 10.

Samsung Galaxy S20 Plus used in the study had a battery capacity of 4,500 milliamp-hour (mAh). When KedyAuth was enabled without retraining, the median battery consumption rate was 0.42 percent (with 0.24 and 0.43 percent for the 2nd and 3rd quartiles, respectively). With retraining enabled, the median battery consumption rate slightly increased to 0.45 percent (with 0.40 and 0.54 percent for the 2nd and 3rd quartiles, respectively),

Table 6. Demographics of the overhead measurement study participants.

Gender		
Female	5	(55%)
Male	4	(45%)
Age		
18–29	2	(22%)
30–39	5	(56%)
40–49	2	(22%)
50–59	0	(0%)
60 and more	0	(0%)
Education		
Less than high school	0	(0%)
High school	0	(0%)
College/University	5	(55%)
Master's/Doctoral degree	4	(45%)
Other	0	(0%)
Occupation		
Computer and Mathematical Occupations	8	(89%)
Office and Administrative Support Occupations	1	(11%)

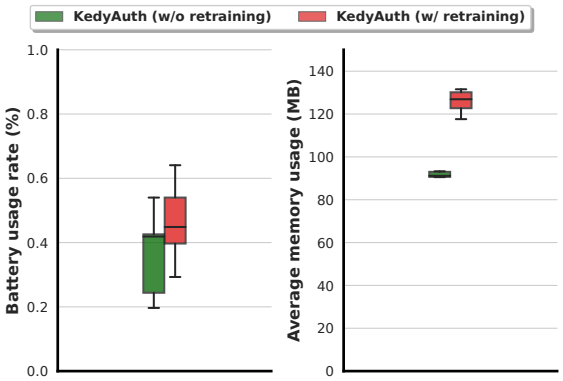


Fig. 10. Median battery and memory usage rates. Green plot represents the second KedyAuth configuration with retraining disabled. Red plot represents the third KedyAuth configuration with retraining enabled.

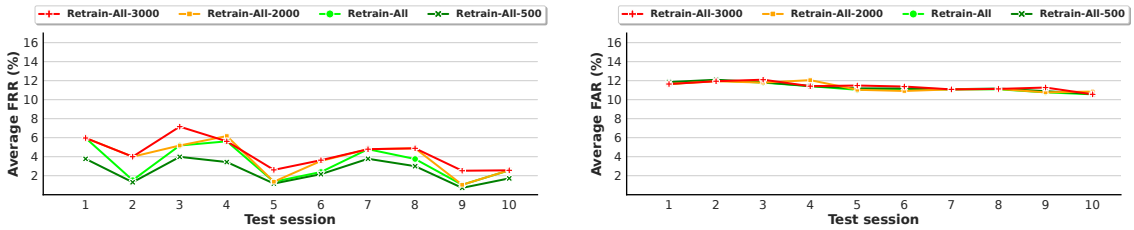
indicating the additional battery consumed from retraining would be insignificant. All nine participants underwent three rounds of retraining.

To measure the memory usage rate we first computed the average usage rate for each typing task based on the memory status logs collected every 10 seconds. We then show the median of those average rates collected across all participants and typing tasks in Figure 10. The median rate without retraining was approximately 91 megabytes across all participants; with retraining enabled, KedyAuth consumed 127 megabytes of memory. Since mainstream smartphones today have several gigabytes of memory available (Samsung Galaxy S20 Plus, for example, had 8 gigabytes of RAM), requiring about 127 megabytes of additional memory (about 1.6 percent for Samsung Galaxy S20 Plus) would be a reasonable overhead.

5.4 Experiencing Keyboard Lags

Six out of nine participants mentioned they did not experience any keyboard lag. Between the first and second configurations (KedyAuth enabled without retraining), two participants experienced 2–3 occurrences of keyboard lags but mentioned that the lags were barely perceptible. One participant experienced more than 5 keyboard lags, again, explaining they were barely perceptible. Between the first and third configurations (retraining enabled), two participants experienced 1–2 occurrences of keyboard lags, explaining they were barely or weakly perceptible.

Taken together, these results indicate the participants' typing experience with KedyAuth was positive: six participants did not experience any lag in all KedyAuth-enabled configurations. All three participants who did experience keyboard lags, explained those lags were barely or weakly perceptible. In Section 6.2, we explain how retraining frequency can be configured based on business requirements to further minimize retraining overheads, and improve users' typing experiences.



(a) Average FRRs (b) Average FARs

Fig. 11. Average FRRs and FARs measured while varying retraining frequency between every 500, 1,000, 2,000, and 3,000 samples. Note, Retrain-All strategy represents a retraining frequency of every 1,000 samples.

6 DISCUSSION

6.1 Setup

Looking at the first EER results, at least 3,000 or more keystroke samples (covering both walking and still contexts) would be needed to train a blender that is capable of achieving about 6.6 percent EER. We imagine that about 1,000 samples could be collected manually during the initial setup, asking users to type a few random sentences with mixed contexts like walking, sitting, and leaning – this initial setup would take about 10 minutes (see Section 4.9). Our unknown context analysis (see Section 4.7) showed that it is integral to cover representative typing postures during setup. An initial blender could be trained at this stage and used with a low threshold value. Once more data becomes available, we would quickly train another blender (e.g., with 3,000 samples) to build a more robust model.

6.2 Retraining Strategies and Managing Overheads

Our retraining strategy analysis showed that higher accuracy can be maintained by training models more frequently. To mitigate untraining effects (see Section 4.7), representative usage contexts need to be covered in the retrain sets in a balanced manner: we suggest (1) always including the initial setup samples (e.g., to make up about 50 percent of the entire set), and (2) store newly collected samples with context labels, and select them in a context-balanced manner during retraining.

Our retraining overhead analysis (see Section 5) showed KedyAuth, configured to retrain the full LightGBM blender every 500 keystrokes, would only consume about 0.45 percent battery per day for a typical user (typing about 1,500 characters on a daily basis). Heavy users (e.g., the most active user from our dataset typed about 3,000 characters per day) could end up using up to about one percent of battery per day. Each retraining would take about 7 seconds to complete. Further, KedyAuth would consume about 130 megabytes of additional memory while users type. Most participants did not notice any keyboard lags while typing, and for those who did notice one or two lags explained they were barely perceptible. Taken together, these results suggest that the continuous prediction and retraining overheads would be manageable, and users feel positive about overall typing experiences.

For vendors who prioritize minimizing retraining overheads, we suggest training models less frequently, and compromise small percentage of authentication accuracy. Figure 11 shows these trade-offs: we measure FRRs and FARs while changing the retraining frequency between every 500 (Retrain-All-500), 1,000 (Retrain-All), 2,000 (Retrain-All-3000), and 3,000 (Retrain-All-3000) samples. Since most users would charge their phones at least once a day (e.g., before going to bed), we believe retraining every 2,000 or 3,000 new samples would have minimal impact on overall battery drain experiences. By configuring KedyAuth to retrain every 3,000 samples, we would

look at about 1.9 percent increase in FRRs, and 0.1 percent increase in FARs (compared to the recommended Retrain-All-500 configuration).

6.3 Facilitating Security/Usability Mode with Threshold Adjustments

Our real-world FAR/FRR trade-off analyses showed that by using threshold values that would achieve, for instance, about 10 percent FRR, many users would benefit from 96 percent or above attack detection rates. These results indicate that attacks described in Section 2.2.3, such as searching for private text messages or using finance/banking apps, can be effectively mitigated while touchscreen keyboards are being used. If a business requires minimizing usability compromises, we could consider using a relaxed 0.4 threshold to guarantee about 73 percent attack detection rate over time (with 0.8 percent FRR) – this would still be a significant improvement from zero continuous authentication.

6.4 Attack Detection Delay

Considering the FRR tradeoffs shown in Figure 7, we could consider using just 10 keystrokes initially to make the first authentication decision quickly. We would then increase the rolling window size in steps of one keystroke (i.e., 11, 12, and so on) until we reach a larger buffer size, which could be, 50 or 70 keystrokes. Once we reach a full buffer size, we would use a "sliding window" to continuously predict scores with every additional keystroke – indicating that as soon as the initial 10 keystroke scores become available, all subsequent decisions can be made with just one additional keystroke. This design would be capable of detecting short malicious SMS text messages being sent – an average English text message contains about 17.2 words [44], and an average English word contains about 5 characters [26]. Furthermore, on average, Russian/Ukrainian sentences contain about 9 words, and each word contains about 5-7 characters [37, 46].

6.5 Multi-factor Authentication

Another scheme that can be considered for continuous authentication is face recognition: users' face can be checked periodically while phones are being used. We believe face recognition and KedyAuth can be used as complementary solutions. Face factor alone will encounter situations where face information is not fully available to make authentication decisions: e.g., people could be wearing masks. The two factors can be used together or interchangeably to bridge such gaps.

One possible attack involves unlocking a stolen device (with a compromised PIN or pattern), finding settings, and simply turning off KedyAuth. Face recognition could also be used to mitigate this type of attack – requesting a second-factor face verification (i.e., a method different from PIN or pattern) to change KedyAuth settings. If an insider finds a subtle way to continuously train his or her own typing behavior (e.g., to annoy the victim), the victim may experience more false rejections over time – in this scenario, the victim would have to reset KedyAuth; again, only the victim should be able to reset it through face (second-factor) verification.

6.6 Limitations

One sampling limitation in our real-world dataset is that all participants were white. As such, the performance of KedyAuth on other ethnic groups remains unknown, and needs to be investigated as a part of future work. Most of the participants had Master's or Doctorate degrees. However, we did not explore the use of words or sentences as classification features, and believe education level would have had minimal impact on the reported accuracy. Most participants typed in Russian and Ukrainian (some also used English), and it is difficult to generalize the findings to different languages. Unlike prior work [3], however, we did not use key-codes, letters, or words as features, and tried to minimize language dependencies. There would be practicality issues if a user is expected to type in multiple languages (and train multiple classifiers) during the setup process: ideally, a classifier trained

based on one language should be compatible with other languages used by the same user. As a preliminary evaluation, based on two participants who had a sufficient number of English keystroke samples, we performed cross-training evaluations between two languages (see Section 4.8) – demonstrating that KedyAuth might be robust in multi-language environments provided that the keyboard layout is the same.

The participants were required to use a fixed "QWERTY" keyboard layout throughout the study. People in Ukraine and Russia commonly use QWERTY keyboards to type Ukrainian, Russian, and English, toggling between languages: all of the participants were familiar with the QWERTY layout. Using a vastly different keyboard layout would probably affect accuracy, and device owners using multiple layouts may have to set up multiple models; or simply turning on "security mode" could disable the use of all other keyboard layouts that are not being protected with KedyAuth. Without further study, however, it is difficult to gauge how typing behaviors and overall accuracy would be affected by keyboard layout changes.

The participants were given new phones to use but they were required to use them for two months. Hence, our dataset captures both short-term and long-term typing behaviors – presenting opportunities to study how KedyAuth would perform when people buy new phones, and when they get used to new keyboards and their typing behaviors change over time. Another limitation is the relatively small number of participants (N=40) – we tried to mitigate this limitation through the additional use of the HMOG dataset (N=100).

Although the overall typing experiences (with retraining enabled) looked positive, several participants did experience small occurrences of keyboard lags, explaining they were barely or weakly perceptible. One possible approach to further optimize retraining overheads, and completely eliminate keyboard lags, is to find a random or systematic sampling method that would allow classifiers to be trained with less number of samples while providing similar level of long-term performance. We recommend that future work seeks to evaluate various sampling methods.

7 RELATED WORK

As smartphones have become increasingly popular, authentication schemes using users' keystroke behaviors on a smartphone's touchscreen (e.g., [8, 15, 36, 38, 41]) have also been extensively studied. Buschek *et al.* [8] considered both temporal and spatial features and achieved EERs ranging from 13.7 to 24.5 percent with data collected from three different hand postures while typing passwords. Sitová *et al.* [41] considered hand-movements, orientation, and grasp (HMOG) as features under walking and sitting contexts. To optimize the model accuracy, 13 and 17 features were used for walking and sitting contexts, respectively. Their fusion method achieved 7.2 and 10.1 percent EERs, respectively, for those two contexts. Crawford and Ahmadzadeh [15] also considered sitting, standing, and walking contexts. Their proposed approach inferred user contexts using gyroscope data with an area under the curve (AUC) of over 90 percent and classified users' typing patterns, which resulted in an AUC of over 93 percent. Saini *et al.* [36] built a system that verifies password-specific typing behaviors, considering both users' typing contexts and phone orientations. They built a model using keystroke and phone movement features and achieved EERs ranging from 2.9 to 5.7 percent for three different user contexts and two different phone orientations. Similarly, Zhang et al. [50] and Shi et al. [39] explored the use of touch and swipe features to enable swiping behavior-based authentication specific to the use of screen lock patterns. Similarly, Shen *et al.* [38] used the motion-sensor data related to touch events and trained a separate HMM model for each phone context ("hand-hold-still", "table-hold-still" or "hand-hold-walk"). For the hand-hold-still context, the constructed model achieved 4 percent FAR and 5 percent FRR. However, detecting such contexts with high accuracy and small delay is also challenging; these context detectors would introduce their own error rates. To that end, we trained a single classifier and demonstrated consistent accuracy ranges across varying real-world contexts and environments.

Centeno *et al.* [9] train user-specific autoencoders based on accelerometer sensor data, and use them as anomaly detectors – demonstrating 4.5 percent EER on the HMOG dataset, and 2.2 percent EER on a real-world dataset; their test settings, however, include all touch (swiping, typing, scrolling, and tapping) events. Abuhamad *et al.* [1] train Long Short-Term Memory (LSTM) models on servers to continuously authenticate entire use of smartphones (i.e., not specific to keyboard use); motion and magnetometer sensor data need to be shared with servers for model training. Both solutions would not satisfy our *"on-device training" requirement*, which states that deep learning models should not be trained on devices (transfer learning is preferred), and sensor data should not be shared with servers. Deb *et al.* [16] do consider those constraints, and explore the use of pre-trained Siamese LSTM models (for feature extraction), again, in the context of authenticating the entire use of smartphones. Their keyboard use specific evaluations show 59 percent true acceptance rate at 0.1 percent FAR. Furthermore, prior literature have not investigated long-term effects, and it is unknown as to how their systems would behave with respect to changing typing behaviors and contexts. Table 8 (see Appendix C) highlights the key differences between KedyAuth and those existing solutions.

We note that it is hard to compare the performance of KedyAuth with all previous studies in a fair manner because their results were mainly evaluated with their own dataset and under different experimental settings. Table 8 presents accuracy results for keystroke dynamics-based authentication solutions including KedyAuth. Hence, to facilitate direct performance comparisons, we used the popularly used HMOG dataset to compare KedyAuth against the top-performing state-of-the-art solutions such as [41]. Our EER evaluation results show clear superiority. Shen *et al.* [38] achieved 5.9 percent HTER on the "walking" set of the HMOG dataset. They applied a wavelet-based denoising method [40] to mitigate the effect of non-stationary noise in sensor signals. In comparison, we achieved 2.3 percent EER on the same (walking) set.

Existing literature do not consider real-world deployment requirements described in Section 2.3. Most studies focus on presenting one-time EERs without considering data storage size, on-device training, and retraining challenges. Our real-world analysis revealed many challenges related to changing typing behaviors and keeping long-term FRRs low. We believe it is necessary to retrain classifiers periodically to help stabilize FRRs over time. Only a few studies [11, 30, 49] investigated the need for adaptive approaches to mitigate long-term error rate instability issues. Xu *et al.* [49] briefly explained that models would have to be updated with new samples based on observations from three participants. Palaskar *et al.* [30] showed that the performance of classifiers trained on users' swiping behaviors could deteriorate over time, and model retraining with the latest data is necessary – such observations are consistent with our findings. However, they focused on swiping and scrolling dynamics, and reporting EER results without threshold value (and FAR/FRR) considerations. In fact, the majority of existing literature does not study the effects of fixing a threshold value – this would be necessary for real-world deployments – and how the overall usability (FRR) and security (FAR) may be affected differently. Chauhan *et al.* [11] investigated authentication modalities related to gait, breathing patterns, and electromyography, again, demonstrating that users' behaviors may change over time, and incremental behavior adaptation is necessary.

Egelman *et al.* [17] conducted an interview study to investigate perceived smartphone data compromise threats and reasons for using locking mechanisms: people were generally concerned with online identity theft, impersonation (e.g., posting on Facebook), and privacy (e.g., looking at personal photos). We specifically focused on identifying threats that involve the use of keyboards: the participants were commonly concerned with adversaries searching for private conversations and text messages.

8 CONCLUSION

Deploying keystroke dynamics-based authentication solutions on smartphones faces two key challenges: (1) error rates need to be maintained throughout long-term use, and (2) feature extraction and model training need to be performed fully on smartphones. Our "Decoder-DTW" and "Correlation" feature sets have been

designed to facilitate on-device feature extraction: Decoder-DTW measures DTW differences between pre-trained autoencoder inputs and outputs, and uses these measurements as features in a novel transfer learning technique. By combining those two feature sets, we significantly outperform state-of-the-art solutions. Our real-world data analysis demonstrated that FRRs can deteriorate over time, and model retraining is necessary to maintain consistently low error rates. Our retraining configuration, which involves retraining the full blender every 500 new samples, ensures about 2.5 percent FRR and 89 percent attack detection rate throughout long term use. Based on this configuration, turning on both continuous authentication and retraining features would consume about 0.45 percent of battery per day for average users. Each retraining would take about 7 seconds to complete. Threshold values can be adjusted based on business needs: e.g., a user-friendly mode can be enabled by using a 0.4 threshold, which would still guarantee about 73 percent attack detection rate (FRR would be around 0.8 percent). The proposed features and retraining techniques have been deployed on the latest Samsung Galaxy series to enable continuous authentication.

REFERENCES

[1] Mohammed Abuhamad, Tamer Abuhmed, David Mohaisen, and DaeHun Nyang. 2020. AUToSen: Deep-Learning-Based Implicit Continuous Authentication Using Smartphone Sensors. *IEEE Internet of Things Journal* 7, 6 (2020).

[2] Sara Amini, Vahid Noroozi, Amit Pande, Satyajit Gupte, Philip S. Yu, and Chris Kanich. 2018. DeepAuth: A Framework for Continuous User Re-Authentication in Mobile Apps. In *Proceedings of the 27th ACM International Conference on Information and Knowledge Management (CIKM)*.

[3] Amith K. Belman and Vir V. Phoha. 2020. Discriminative Power of Typing Features on Desktops, Tablets, and Phones for User Identification. *ACM Transactions on Privacy and Security* 23, 1 (2020).

[4] J. Bergstra, D. Yamins, and D. D. Cox. 2013. Making a Science of Model Search: Hyperparameter Optimization in Hundreds of Dimensions for Vision Architectures. In *Proceedings of the 30th International Conference on International Conference on Machine Learning (ICML)*.

[5] Donald J. Berndt and James Clifford. 1994. Using Dynamic Time Warping to Find Patterns in Time Series. In *Proceedings of the 3rd International Conference on Knowledge Discovery and Data Mining (AAAIWS)*.

[6] Joseph Bonneau, Sören Preibusch, and Ross J. Anderson. 2012. A Birthday Present Every Eleven Wallets? The Security of Customer-Chosen Banking PINs. In *Proceedings of the 16th International Conference on Financial Cryptography and Data Security (FC)*.

[7] Daniel Buschek, Benjamin Bisinger, and Florian Alt. 2018. ResearchIME: A Mobile Keyboard Application for Studying Free Typing Behaviour in the Wild. In *Proceedings of the 36th Annual ACM Conference on Human Factors in Computing Systems (CHI)*.

[8] Daniel Buschek, Alexander De Luca, and Florian Alt. 2015. Improving Accuracy, Applicability and Usability of Keystroke Biometrics on Mobile Touchscreen Devices. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems (CHI)*.

[9] M. P. Centeno, A. v. Moorsel, and S. Castruccio. 2017. Smartphone Continuous Authentication Using Deep Learning Autoencoders. In *Proceedings of the 15th Annual Conference on Privacy, Security and Trust (PST)*.

[10] S. Cha, S. Kwag, H. Kim, and J. H. Huh. 2017. Boosting the Guessing Attack Performance on Android Lock Patterns with Smudge Attacks. In *Proceedings of the 12th ACM on Asia Conference on Computer and Communications Security (ASIACCS)*.

[11] Jagmohan Chauhan, Young D. Kwon, Pan Hui, and Cecilia Mascolo. 2020. ContAuth: Continual Learning Framework for Behavioral-Based User Authentication. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* 4, 4 (2020).

[12] Zhaomin Chen, Chai Kiat Yeo, Bu Sung Lee, and Chiew Tong Lau. 2018. Autoencoder-based network anomaly detection. In *2018 Wireless Telecommunications Symposium (WTS)*. 1–5. https://doi.org/10.1109/WTS.2018.8363930

[13] G. Cho, J. H. Huh, J. Cho, S. Oh, Y. Song, and H. Kim. 2017. SysPal: System-Guided Pattern Locks for Android. In *Proceedings of the 38th IEEE Symposium on Security and Privacy (S&P)*.

[14] Francois Chollet et al. 2015. *Keras*. https://github.com/fchollet/keras

[15] Heather Crawford and Ebad Ahmadzadeh. 2017. Authentication on the Go: Assessing the Effect of Movement on Mobile Device Keystroke Dynamics. In *Proceedings of the 13th Symposium on Usable Privacy and Security (SOUPS)*.

[16] Debayan Deb, Arun Ross, Anil K. Jain, Kwaku Prakah-Asante, and K. Venkatesh Prasad. 2019. Actions Speak Louder Than (Pass)words: Passive Authentication of Smartphone Users via Deep Temporal Features. In *Proceeding of 12th IAPR International Conference on Biometrics (ICB)*.

[17] Serge Egelman, Sakshi Jain, Rebecca S Portnoff, Kerwell Liao, Sunny Consolvo, and David Wagner. 2014. Are You Ready to Lock?. In *Proceedings of the 21st ACM SIGSAC conference on Computer and communications security (CCS)*.

[18] European Banking Authority. 2019. *EBA publishes an Opinion on the elements of strong customer authentication under PSD2*. Retrieved June 21, 2019 from https://www.eba.europa.eu/eba-publishes-an-opinion-on-the-elements-of-strong-customer-authentication-under-psd2

[19] Hassan Ismail Fawaz, Germain Forestier, Jonathan Weber, Lhassane Idoumghar, and Pierre-Alain Muller. 2019. Deep learning for time series classification: a review. *Data Mining and Knowledge Discovery* 33 (2019).
[20] Joseph L Fleiss, Bruce Levin, and Myunghee Cho Paik. 2013. *Statistical methods for rates and proportions*. John Wiley & Sons.
[21] T. Giorgino. 2009. Computing and Visualizing Dynamic Time Warping Alignments in R: The dtw Package. *Journal of Statistical Software* 31 (2009).
[22] Ian J. Goodfellow, Yoshua Bengio, and Aaron Courville. 2016. *Deep Learning*. MIT Press.
[23] M. Harbach, A. D. Luca, and Serge Egelman. 2016. The Anatomy of Smartphone Unlocking: A Field Study of Android Lock Screens. In *Proceedings of the 34th Annual ACM Conference on Human Factors in Computing Systems (CHI)*.
[24] Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. 2017. LightGBM: A Highly Efficient Gradient Boosting Decision Tree. In *Proceedings of the 31st International Conference on Neural Information Processing Systems (NIPS)*.
[25] Hassan Khan, Urs Hengartner, and Daniel Vogel. 2015. Usability and Security Perceptions of Implicit Authentication: Convenient, Secure, Sometimes Annoying. In *Proceedings of the 11th Symposium On Usable Privacy and Security (SOUPS)*.
[26] I. Scott MacKenzie and Kumiko Tanaka-Ishii. 2007. *Text Entry Systems: Mobility, Accessibility, Universality*. Morgan Kaufmann Publishers Inc.
[27] Kathleen Macqueen, Eleanor McLellan-Lemal, K. Bartholow, and B. Milstein. 2008. Team-based codebook development: Structure, process, and agreement. *Handbook for team-based qualitative research* (2008).
[28] Philipp Markert, Daniel V. Bailey, Maximilian Golla, Markus Dürmuth, and Adam J. Aviv. 2020. This PIN Can Be Easily Guessed: Analyzing the Security of Smartphone Unlock PINs. In *Proceedings of the 41st IEEE Symposium on Security and Privacy (S&P)*.
[29] Yisroel Mirsky, Tomer Doitshman, Yuval Elovici, and Asaf Shabtai. 2018. Kitsune: an ensemble of autoencoders for online network intrusion detection. *arXiv preprint arXiv:1802.09089* (2018).
[30] N. Palaskar, Z. Syed, S. Banerjee, and C. Tang. 2016. Empirical Techniques to Detect and Mitigate the Effects of Irrevocably Evolving User Profiles in Touch-Based Authentication Systems. In *Proceedings of the 17th IEEE International Symposium on High Assurance Systems Engineering (HASE)*.
[31] Sinno Jialin Pan and Qiang Yang. 2010. A Survey on Transfer Learning. *IEEE Transactions on Knowledge and Data Engineering* 22, 10 (2010), 1345–1359. https://doi.org/10.1109/TKDE.2009.191
[32] Vishal M. Patel, Rama Chellappa, Deepak Chandra, and Brandon Barbello. 2016. Continuous User Authentication on Mobile Devices: Recent progress and remaining challenges. *IEEE Signal Processing Magazine* 33, 4 (2016), 49–61. https://doi.org/10.1109/MSP.2016.2555335
[33] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research* 12 (2011).
[34] Android Developers Portal. n.d.. *Android dumpsys tool*. https://developer.android.com/studio/command-line/dumpsys?hl=en
[35] Google Developers Portal. 2020. *Activity Recognition Transition API Codelab*. Retrieved September 16, 2020 from https://codelabs.developers.google.com/codelabs/activity-recognition-transition/index.html#6
[36] Baljit Singh Saini, Parminder Singh, Anand Nayyar, Navdeep Kaur, Kamaljit Singh Bhatia, Shaker El-Sappagh, and Jong-Wan Hu. 2020. A Three-Step Authentication Model for Mobile Phone User Using Keystroke Dynamics. *IEEE Access* 8, 3 (2020).
[37] Irina Sekerina, Anna Laurinavichyute, Svetlana Alexeeva, Kristina Bagdasaryan, and Reinhold Kliegl. 2019. Russian Sentence Corpus: Benchmark measures of eye movements in reading in Russian. *Behavior Research Methods* 51 (2019).
[38] Chao Shen, Y. Li, Yufei Chen, X. Guan, and R. Maxion. 2018. Performance Analysis of Multi-Motion Sensor Behavior for Active Smartphone Authentication. *IEEE Transactions on Information Forensics and Security* 13 (2018).
[39] Dai Shi, Dan Tao, Jiangtao Wang, Muyan Yao, Zhibo Wang, Houjin Chen, and Sumi Helal. 2021. Fine-Grained and Context-Aware Behavioral Biometrics for Pattern Lock on Smartphones. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* 5, 1 (2021).
[40] W. Shi, J. Yang, Yifei Jiang, Feng Yang, and Yingen Xiong. 2011. SenGuard: Passive user identification on smartphones using multiple sensors. In *Proceedings of the 7th IEEE International Conference on Wireless and Mobile Computing, Networking and Communications (WiMob)*.
[41] Z. Sitová, J. Šeděnka, Q. Yang, G. Peng, G. Zhou, P. Gasti, and K. S. Balagani. 2016. HMOG: New Behavioral Biometric Features for Continuous Authentication of Smartphone Users. *IEEE Transactions on Information Forensics and Security* 11, 5 (2016).
[42] Jess Solano, Lizzy Tengana, Alejandra Castelblanco, Esteban Rivera, Christian E. Lopez, and Martín Ochoa. 2020. A Few-Shot Practical Behavioral Biometrics Model for Login Authentication in Web Applications. In *Proceedings of Workshop on Measurements, Attacks, and Defenses for the Web (MADWeb)*.
[43] Shridatt Sugrim, Can Liu, Meghan McLean, and Janne Lindqvist. 2019. Robust Performance Metrics for Authentication Systems. In *Proceedings of the 26th Annual Network and Distributed System Security Symposium (NDSS)*.
[44] Caroline Tagg. 2009. A corpus linguistics study of SMS text messaging.

[45] PEW Research Center. 2019. *Mobile Technology and Home Broadband 2019.* https://www.pewresearch.org/internet/2019/06/13/mobile-technology-and-home-broadband-2019/
[46] Stefan Trost Media. n.d.. *Character Frequencies.* https://www.sttmedia.com/characterfrequencies
[47] Michael Tschannen, Olivier Bachem, and Mario Lucic. 2018. Recent Advances in Autoencoder-Based Representation Learning. *CoRR* abs/1812.05069 (2018). arXiv:1812.05069 http://arxiv.org/abs/1812.05069
[48] Sebastian Uellenbeck, Markus Dürmuth, Christopher Wolf, and Thorsten Holz. 2013. Quantifying the security of graphical passwords: the case of android unlock patterns. In *Proceedings of the 20th ACM SIGSAC conference on Computer and Communications Security (CCS).*
[49] Hui Xu, Yangfan Zhou, and Michael R. Lyu. 2014. Towards Continuous and Passive Authentication via Touch Biometrics: An Experimental Study on Smartphones. In *Proceedings of the 10th Symposium On Usable Privacy and Security (SOUPS).*
[50] Xinchen Zhang, Yafeng Yin, Lei Xie, Hao Zhang, Zefan Ge, and Sanglu Lu. 2020. TouchID: User Authentication on Mobile Devices via Inertial-Touch Gesture Analysis. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies* 4, 4 (2020).

A ONE-DIMENSION DYNAMIC TIME WARPING

1D DTW measures the similarity between specific regions of two given time series data [5]. Given the two time series, "original input" $X = (x_1, x_2, ..., x_N)$ and "reconstructed output" $X' = (x'_1, x'_2, ..., x'_M)$, the goal of DTW is to find the best mapping from elements in X to those in X' such that the overall sum of distances between them $d(x_n, x'_m)$ is minimized. We use Euclidean distance to measure the distance $d(\cdot, \cdot)$. N and M are the sizes of the time series. In our case, N and M are equal to 72, and we denote this value as L. Then, the warping path can be obtained as $W = [(wx(1), wx'(1)), ..., (wx(L), wx'(L))]$ consisting of elements denoted as $wx(l)$ and $wx'(l)$, which represent the indices of related time series data points. The final $DTW(X, X')$ equation can be defined based on W to minimize the overall sum of distances between the elements in X and X':

$$DTW(X, X') = \arg\min_{W} \sum_{l=1}^{L} d\left(X_{wx(l)}, X'_{wx'(l)}\right) \tag{1}$$

DTW matching is constrained by [5] (1) "continuity" where every index from X must be matched with one or more indices from X', and vise versa, (2) "boundaries" where the first and last indices from X must be matched with the first and last indices from X', and (3) "monotonicity" where the mapping of the indices from X to the indices from X' must be monotonically increasing, and vise versa.

B REAL-WORLD DATA DEMOGRAPHICS

We recruited 40 participants in total. Table 7 shows the details of the demographics. All participants were white, and the majority were in the 18–29 and 30–39 age groups (42.5% and 40%) respectively. 52.5% were female. 77.5% had Master's or Doctorate degrees, and 12.5% had a high school diploma. 32 (80%) participants said they unlock their phones a few times per hour. 6 (15%) participants said they unlock their phones once per hour. 20 (50%) participants said they use their phones 4 to 6 hours per day. 19 (47.5%) participants said they use their phones for 2 to 4 hours per day. Additionally, 19 (47.5%) participants said they used their phones for more than 5 years. 8 (20%) participants said they used their phones for 1 to 2 years. As for the phone models, the majority of the participants, 25%, 22.5%, and 20% were using Xiaomi, Samsung, and Meizu phones, respectively. Only one participant was using an iPhone: this is because iPhones are not as popular in Ukraine.

C COMPARING KEDYAUTH PERFORMANCE WITH STATE-OF-THE-ART SOLUTIONS

Table 8 provides a summary of the performance of state-of-the-art keystroke dynamics-based authentication solutions, and compares their EER performance against KedyAuth accuracy results.

Table 7. Demographics of the real-world data collection study participants.

Gender		
Female	21	(52.5%)
Male	19	(47.5%)
Age		
18 - 29	17	(42.5%)
30 - 39	16	(40.0%)
40 - 49	5	(12.5%)
50 and more	2	(5.0%)
Education		
Less then high school	0	(0.0%)
High school	5	(12.5%)
College/University	4	(10.0%)
Master's/Doctoral degree	31	(77.5%)
Occupation		
Architecture and Engineering Occupations	3	(7.5%)
Arts, Design, Entertainment, Sports, and Media Occupations	2	(5.0%)
Business and Financial Operations Occupations	5	(12.5%)
Computer and Mathematical Occupations	3	(7.5%)
Education, Training, and Library Occupations	2	(5.0%)
Healthcare Practitioners and Technical Occupations	1	(2.5%)
Management Occupations	4	(10.0%)
Out of Work	3	(7.5%)
Sales and Related Occupations	3	(7.5%)
Students	1	(2.5%)
Legal Occupations	5	(12.5%)
Transportation and Materials Moving Occupations	1	(2.5%)
Life, Physical, and Social Science Occupations	3	(7.5%)
Others	4	(10.0%)
Ethnicity		
White	40	(100.0%)
Black	0	(0.0%)
Asian	0	(0.0%)
Others	0	(0.0%)
Unlock trials		
A few times per hour	32	(80.0%)
Once per hour	6	(15.0%)
A few times a day	1	(2.5%)
Once a year	0	(0.0%)
No idea	1	(2.5%)
Average phone usage time per day (min)		
0 - 119	0	(0.0%)
120 - 239	19	(47.5%)
240 - 360	20	(50.0%)
360+	1	(2.5%)
Phone used period		
Less than a year	3	(7.5%)
1 - 2 years	8	(20.0%)
2 - 3 years	3	(7.5%)
3 - 4 years	3	(7.5%)
4 - 5 years	4	(10.0%)
Over 5 years	19	(47.5%)
Phone model actually used by the participants		
Samsung	9	(22.5%)
Apple	1	(2.5%)
Meizu	8	(20.0%)
Xiaomi	10	(25.0%)
Huawei	6	(15.0%)
Lenovo	2	(5.0%)
Other	4	(10.0%)

Table 8. Comparing the KedyAuth performance against state-of-the-art keystroke dynamics based authentication solutions. "Free Text" indicates whether the proposed system authenticates/verifies all text being entered. "On-device?" indicates whether the proposed system can be fully trained on smartphones, and satisfies our on-device training requirement. Note, "Performance" is measured on different datasets, and sometimes more than one authentication modalities (e.g., typing, swiping, tapping, or gait) may have been evaluated, and should not be compared directly. HMOG dataset, however, has been used for evaluation in two other systems.

Method	Dataset	# Subjects	Study duration	Free text	Classifier	On-device?	Performance
KedyAuth	HMOG	100	-	✔	SVM Ensemble	✔	4.91% EER
	HMOG	100	-	✔	LGBM Ensemble	✔	6.57% EER
	Real-world study (long term)	40	**2 Months**	✔	SVM Ensemble	✔	2.03% FRR, 11.44% FAR
	Real-world study (long term)	40	**2 Months**	✔	LGBM Ensemble	✔	2.51% FRR, 11.34% FAR
Sitová *et al.* [41]	HMOG	100	-	✔	Scaled Manhattan	✔	7.16–10.05% EER
Shen *et al.* [38]	HMOG (Walking)	100	-	✔	Hidden Markov Model	✔	5.9% HTER (*KedyAuth*: 2.3% EER)
	Lab study	102	Several days	✔	Hidden Markov Model	✔	5.03% FRR, 3.98% FAR
Crawford *et al.* [15]	Lab study	39	-	✔	Logistic Regression	✔	97% AUC
Saini *et al.* [36]	Lab study	40	-	✗ (passwords)	Random Forest	✔	2.9–5.7% EER
Buschek *et al.* [8]	Lab study	28	-	✗ (passwords)	*LSAD	✔	13.74–24.48% EER
Centeno *et al.* [9]	CrowdSignals.io real-world study	20	Several days	✔	Autoencoder	✗	2.2% EER (all behaviors)
Abuhamad *et al.* [1]	Real-world study (short term)	82	5 days	✔	LSTM	✗	6.67% FRR, 0.95% FAR (all behaviors)

*LSAD : Least Squares Anomaly Detection