Towards User Re-Authentication on Mobile Devices via On-Screen Keyboard

Zijiang Hao, Qun Li
College of William and Mary, Williamsburg, VA, USA
{hebo, liqun}@cs.wm.edu

***Abstract*—Mobile devices have become our true companions in recent years. While bringing plenty of convenience, they also come with many security and privacy concerns. Being small, a mobile device is prone to loss or theft. Privacy data such as emails in a saved Gmail or Yahoo account on a lost device can be easily accessed by an unwanted visitor. Therefore, it is essential to research methods protecting mobile devices from any such unauthorized access. In this paper, we explore the potential of re-authenticating mobile device users by exploiting the biometrics derived from their tapping behaviors on the on-screen keyboard. We conduct an offline analysis on a dataset collected from 33 subjects using a Google Nexus S phone. The results show that more than 90% accuracy can be achieved for text input with 20 continuous key-taps.**

I. Introduction

Recent years have seen the fast spread of mobile devices. According to a recent Cisco's white paper [1], the number of mobile devices has far exceeded the world population. Mobile devices are indeed part of people's life nowadays. People use them to send and receive emails (e.g., through Gmail), access social networks (e.g., Facebook and Twitter), enjoy online banking and shopping services, and so on.

However, the increasing use of mobile devices results in the rise of security and privacy concerns. First, a mobile device is prone to loss and theft because it is designed to be small and portable. Symantec claims that one in every three U.S. users has experience with cell phone loss or theft [2]. Second, privacy data such as personal pictures on a lost device can be directly accessed by an unwanted visitor without any protection. Moreover, there are many apps running on mobile devices that allow users to store their accounts and passwords on local disks. Therefore, a lost mobile device with saved account information may cause huge damage to its owner. Consider this scenario: an attacker steals a smartphone and finds that it has Gmail and Facebook installed with saved accounts. He or she may log into Gmail/Facebook, and send advertising or obscene emails/messages, which would seriously hurt the owner's reputation.

Arguably, there are security mechanisms designed for mobile devices to prevent local data from being accessed by attackers. Authentication mechanisms, such as text-based and shape-based passwords and identification numbers, are based on secret knowledge. However, these methods suffer from several problems. First, based on our observation, few users enable such authentication mechanisms on their mobile devices. In contrast, most of them prefer to unlock a screen by sliding a bar (in iOS) or a circle (in Android). Second, even if a PIN or password is set for unlocking a mobile device, there are still many ways to breach this security barrier: (1) PINs with a limited length (e.g., four digits) are prone to brute-force attacks if there is no restriction on trial times. (2) All of these methods suffer from shoulder surfing, eavesdropping and smudge attacks. (3) An attacker can physically access the raw data stored on a mobile device without launching its operating system.

Some mobile devices support remote data wiping, through which the owner of a lost or stolen mobile device can remotely wipe all the data stored on the device [3]. This mechanism is threatened by several situations: (1) An attacker may have already accessed the data before the owner notices the device loss. (2) An attacker may cut down all possible connections to the mobile device so that it cannot be controlled remotely. (3) The owner has no data backup and thus is unwilling to wipe the device.

A natural solution to the aforementioned challenges is to exploit biometrics to re-authenticate the user who is interacting with the mobile device. Unlike secret knowledge-based mechanisms, biometrics are unique for individuals and are difficult to forge. Moreover, many biometrics-based security mechanisms are non-intrusive (i.e., requiring no user intervention), which means that they introduce little negative effect on user experience.

To this end, we present SafeKey, a security system for mobile devices that exploits the biometrics of user tapping behaviors to enforce user re-authentication. More specifically, SafeKey re-authenticates users at runtime, using the biometrics non-intrusively derived from the user tapping behaviors on the on-screen keyboard. Our main contributions are summarized as follows.

- First, we derive several biometrics from user tapping behaviors on the on-screen keyboard and use them to authenticate mobile device users. To be practical, we consider the user authentication problem as a *one-class* classification problem and hence remove the assumption on the availability of non-owners' data in the training phase.
- Second, we design SafeKey, a non-intrusive system that exploits the biometrics of user tapping behaviors on the on-screen keyboard. It constructs a one-class classifier for each key displayed on the keyboard, and combines all the classifiers to authenticate the user who is tapping. Additionally, it adapts the classifiers to the changes of user input behaviors

through classifier re-training.
- Third, we implement a SafeKey prototype on a Google Nexus S smartphone, and collect data from 33 subjects (7 females and 26 males). Evaluation results demonstrate that SafeKey can achieve up to 90% accuracy for user text input with 20 continuous key-taps, and longer input can make the accuracy even higher.

Some existing works [4], [5], [6], [7], [8], [9], [10] also exploit biometrics derived from user behaviors on the touch screen to (re-)authenticate users. These works either derive biometrics from other user behaviors than key-taps, such as sliding gestures, or conduct two- or multi-class classification to enforce user authentication. Our work distinguishes itself by re-authenticating users based on their on-screen keyboard tapping behaviors, and employing one-class classification to remove the assumption on the availability of non-owners' data. It is unreasonable to assume that data from other people (non-owners) is always available in the training phase. Some third parties may be willing to provide such data, but it will be a disaster if some of them are malicious. Even if a third party can be fully trusted, the data it provides may be limited and cannot represent the entire population. Other biometrics, such as fingerprint, finger vein structure and iris structure, are also used to authenticate mobile device users in the literature [11], [12], [13]. Our work is complementary to these works, as we use different biometrics to authenticate users.

The rest of this paper is organized as follows. In Section II, we present the design of SafeKey, with its evaluation presented in Section III. Then we conclude in Section IV.

II. SafeKey Architecture

A. Threat Model

We summarize the threat model of SafeKey below.

I) The attacker can physically access the victim's mobile device for some reason (e.g., the device is stolen by the attacker).

II) The victim has stored his/her account information of an online app (e.g., Gmail or Facebook), so that the attacker can easily log into the app with the victim's account.

III) After logging into the app, the attacker inputs a piece of text by tapping the software keyboard displayed on the touch screen, and sends the text out on behalf of the victim.

In addition to this threat model, SafeKey also makes the following assumptions.

I) The first N_{train} taps on the software keyboard after the mobile device comes out of the factory are considered only from the owner. Each tap after them is considered from either the owner, or a non-owner, which is determined by SafeKey.

II) All taps made in a relatively short time period (e.g., 30 seconds) are considered from the same user (i.e., either the owner, or a non-owner).

We believe that these assumptions are reasonable. For the first one, users always protect new devices with great caution, and use them a lot for curiousness. For the second one, it is very unlikely that multiple users tap the software keyboard alternatively during a short time period.

B. SafeKey Overview

Figure 1 depicts the architecture of SafeKey. There are five components in SafeKey.

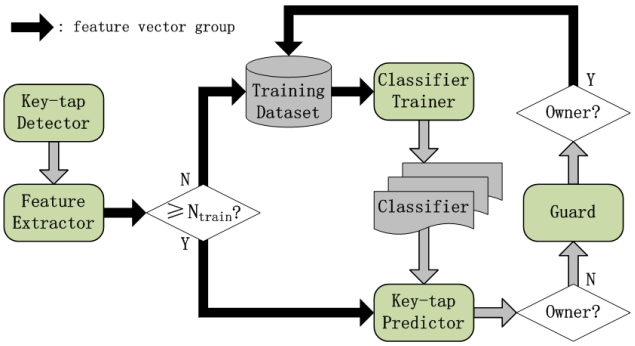


Fig. 1. SafeKey architecture.

1) Key-tap Detector: The Key-tap Detector captures every tap on the software keyboard and records the instantaneous readings of the touch screen, accelerometer, and gyroscope. In our prototype, the Key-tap Detector is implemented as a listener which listens to the software keyboard for motion events. The sensor readings it records for each key-tap includes:

- $keycode$: the code of the tapped key.
- t_{down}: the time of the "key down" event.
- t_{up}: the time of the "key up" event.
- x: the x-axis coordinate on the software keyboard of the "key down" event.
- y: the y-axis coordinate on the software keyboard of the "key down" event.
- $pressure$: the pressure value of the "key down" event.
- $size$: the touched size value of the "key down" event.
- $accl[\,]$: the linear acceleration sensor readings during [t_{down}, t_{up}].
- $gyro[\,]$: the gyroscope readings during [t_{down}, t_{up}].

Key-taps are grouped based on the second assumption made by SafeKey: *all key-taps made during a short time period are from the same user*. In our prototype system, this function is implemented by setting a threshold T_{group} for the time interval between two consecutive key-taps in the same group. When a key-tap K_n is detected, the Key-tap Detector waits for the next key-tap K_{n+1} for T_{group} time, and considers K_n the last key-tap of the current key-tap group if K_{n+1} does not come in T_{group} time. The Key-tap Detector then transfers the recorded data in the unit of key-tap group to the Feature Extractor.

2) Feature Extractor: After receiving the data of a key-tap group, Feature Extractor extracts a feature vector for each key-tap in the group. The features extracted for each key-tap include:

- $keycode$: the code of the tapped key (for indexing purpose).
- $duration$: the time difference between t_{up} and t_{down} of the key-tap.

- x: the raw x-axis coordinate reading of a key-tap.
- y: the raw y-axis coordinate reading of a key-tap.
- $pressure$: the raw $pressure$ reading of a key-tap.
- $size$: the raw $size$ reading of a key-tap.
- avg_{accl}: the average magnitude of all linear acceleration readings during $[t_{down}, t_{up}]$.
- avg_{gyro}: the average magnitude of all angular acceleration readings during $[t_{down}, t_{up}]$.

After feature extraction, the Feature Extractor checks the number of feature vectors that have been collected. If it is less than N_{train}, more feature vectors will be collected. Otherwise, extracted features are transferred to either the Classifier Trainer or the Key-tap Predictor for further processing.

3) Classifier Trainer: The Classifier Trainer trains a one-class classifier for each key covered by the available feature vectors. Unlike the two- and multi-class classifiers adopted by existing works that require training data from both the owner and non-owners, the training algorithm of one-class classifiers only requires data from the owner. In SafeKey, it is guaranteed that the Classifier Trainer can only receive feature vectors from the owner.

The classification algorithm used in our prototype is the one-class Support Vector Machine (SVM). We choose SVM as the classification algorithm because: (1) It is one of the best off-the-shelf classification algorithms and has been widely used in various fields such as human activity recognition and document classification. (2) Its execution time has been demonstrated to be short on mobile devices. In the training phase, the Classifier Trainer divides the training feature vectors into groups based on their $keycode$s and trains a one-class SVM classifier for each group. It is worth noticing that different $keycode$s may be in the same group. For example, 'a' and 'A' are in the same group, since the two $keycode$s are actually mapped to the same key on the software keyboard.

4) Key-tap Predictor: When the Key-tap Predictor receives a group of feature vectors, it determines whether the group belongs to the owner or not in the following way.

I) For each feature vector v in the group, if v's $keycode$ has an existing trained classifier, the Key-tap Predictor feeds v into the corresponding classifier based on v's $keycode$, and gets the prediction result (i.e., whether v belongs to the owner). Otherwise, the Key-tap Predictor does nothing.

II) For the group, if the number of feature vectors classified as positive (i.e., belonging to the owner) is greater than or equal to a pre-defined threshold T_G, the group of feature vectors is considered positive. Otherwise, it is considered negative (i.e., belonging to a non-owner).

When a group of feature vectors is considered negative, the Guard is triggered.

5) Guard: When the Guard is triggered, SafeKey assumes that the mobile device is being attacked as described in Section II-A. To protect the mobile device from being misused, the Guard performs a pre-determined action, which is locking the screen and asking for a secure password to unlock it.

If the user unlocks the screen quickly, SafeKey considers that the user is the owner, and that the Key-tap Predictor has just mis-classified the feature vectors. In this case, the feature vectors are stored. If the number of stored feature vectors reaches a pre-determined threshold (e.g., half of N_{train}), the Classifier Trainer component is triggered again to re-train the existing classifiers for the keys whose $keycode$s are covered by the stored feature vectors. By doing this, SafeKey automatically updates its classifiers to capture the changes of the owner's input patterns. Note that the Classifier Trainer may train new classifiers in this process, for the keys whose $keycode$s have not been covered previously but are covered by the stored feature vectors.

III. Evaluation

In this section, we evaluate SafeKey through data collection and offline analysis, and compare the results of SafeKey with those of other algorithms.

A. Data Collection

We collect a key-tap dataset from 33 subjects, including 7 females and 26 males. All of them are between 22 and 30 years old. Each subject is asked to casually input five text messages on a Google Nexus S smartphone every day, with each text message consisting of at least 80 words. We conduct this experiment for several days to ensure that each subject provides at least 6000 key-taps. The collected dataset is used for further offline analysis on a desktop PC.

When inputting the text messages, each subject is asked to sit down, hold the smartphone by his/her left hand, and tap the software keyboard with his/her right hand. However, different subjects are allowed to have different tapping gestures, two examples of which are illustrated in Figure 2.

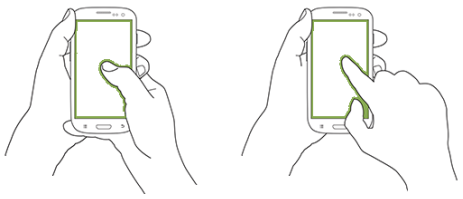
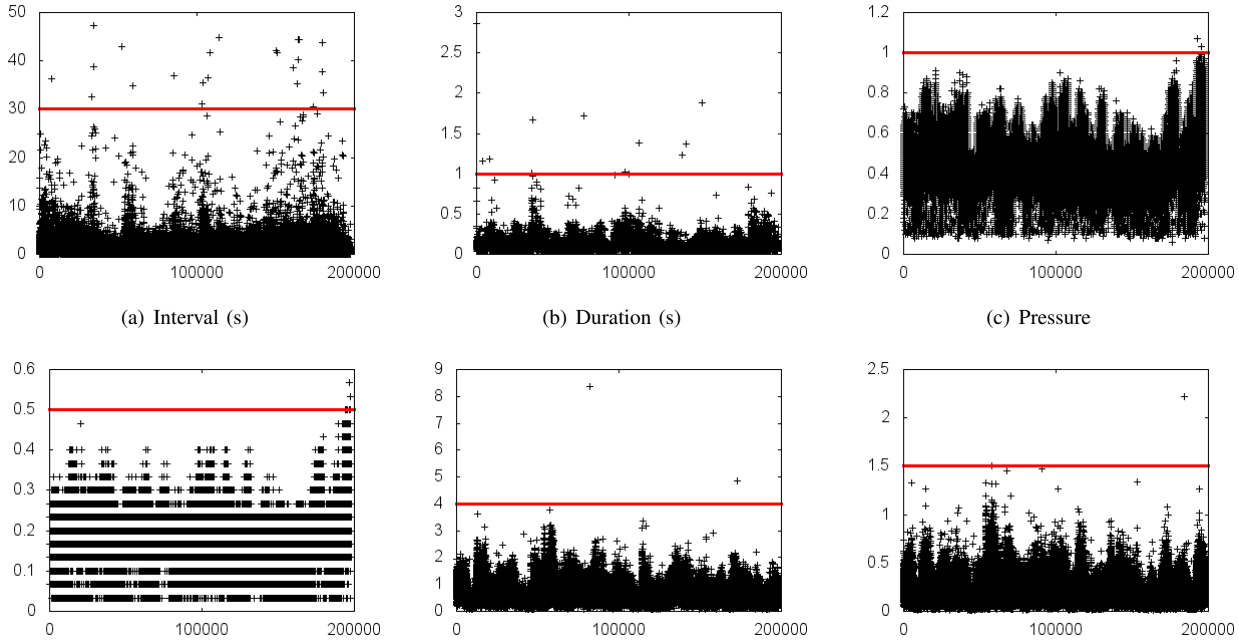


Fig. 2. Possible tapping gestures. The subject holds the smartphone with his/her left hand, and taps the software keyboard with his/her right hand.

B. Outlier Elimination

Before evaluating the performance with the collected dataset, we perform an important preprocessing, i.e., eliminating the outliers in the dataset. Outliers are the data points that do not conform to normal behaviors. They may have great impact on classifier training and result in degraded classifier performance.

There are many ways to detect and eliminate outliers [14]. In the evaluation, we eliminate outliers by suppressing, i.e., we set an upper bound for each feature, and a feature value exceeding this bound is considered an outlier value and will be suppressed to this upper bound. Figure 3 illustrates the values of each feature extracted from all users' data. The horizontal axis depicts different key-taps, while the vertical axis depicts



(a) Interval (s) (b) Duration (s) (c) Pressure

(d) Size (e) Avg. magnitude of linear accl. (m/s^2) (f) Avg. magnitude of angular accl. (rad/s)

Fig. 3. Feature values extracted from the dataset.

the feature values. The red line in each subfigure illustrates the upper bound of the corresponding feature.

Intervals illustrated in Figure 3(a) (i.e., the time difference between a current key-tap's t_{down} and the previous key-tap's t_{up}) are not used in our prototype system. However, they are used in the evaluation for comparison purpose. For this reason, we also plot them in the figure. Sizes illustrated in Figure 3(d) are discrete values due to the low precision of Google Nexus S's touch screen. Size and pressure readings returned by the Android API are normalized values, which have no units. The units of other features are denoted in the subfigure captions.

C. Evaluation Methods and Performance Metrics

In the evaluation, we plan to measure the performance of SafeKey and those of other methods. In this section, we present how the evaluation is performed and what metrics are used for measuring the performance.

1) Data Division, Training, and Testing: For each user u, we equally divide her/his 6000 key-taps into two sets. The first 3000 key-taps are used for training and denoted as $set_{1,u}$, while the last 3000 key-taps are used for validation and denoted as $set_{2,u}$. The data of each key is proportionally distributed into $set_{1,u}$ and $set_{2,u}$.

For $set_{1,u}$, we further divide it into two sets: the first 2000 key-taps are used for training and denoted as $set_{train1,u}$, while the last 1000 key-taps are used for testing and denoted as $set_{test1,u}$. The data of each key is proportionally distributed into $set_{train1,u}$ and $set_{test1,u}$. In the training phase, we conduct the grid search [15] to investigate different parameter combinations. For each parameter combination, we train a one-class SVM for each key with $set_{train1,u}$ following the 10-fold cross validation routine. For each key, the parameter combination that achieves the highest performance is considered as optimal and selected for that key. The purpose of taking the cross validation routine in the training phase is to alleviate the overfitting problem. In addition, we separate $set_{test1,u}$ from $set_{train1,u}$ in the training dataset because we use $set_{test1,u}$ to optimize any other system parameter other than the classifier parameters. The system parameter will be introduced later in this section.

For $set_{2,u}$, we also divide it into two sets: the first 2000 key-taps are used for training the final classifiers for the keys with the selected optimal parameter combinations and denoted as $set_{train2,u}$, while the last 1000 key-taps are used for testing the trained classifiers and denoted as $set_{test2,u}$. The data of each key is proportionally distributed into $set_{train2,u}$ and $set_{test2,u}$.

For convenience purpose, we introduce the following four terms, which will be used later in this section.

$$set_{train1} = set_{train1,1}, ..., set_{train1,N};$$
$$set_{test1} = set_{test1,1}, ..., set_{test1,N};$$
$$set_{train2} = set_{train2,1}, ..., set_{train2,N};$$
$$set_{test2} = set_{test2,1}, ..., set_{test2,N};$$

, where $N = 33$ is the number of users.

2) Performance Metrics: To comprehensively measure the performance, we select three metrics: *accuracy*, *false acceptance rate* (FAR), and *false rejection rate* (FRR). Their mathematical definitions are listed as follows:

$$accuracy = \frac{TP + TN}{TP + FN + TN + FP}$$

$$FAR = FP \,/\, (TN + FP)$$

$$FRR = FN \,/\, (TP + FN)$$

TABLE I
AUTHENTICATION PERFORMANCE, SAFEKEY

(a) Performance on training dataset $<set_{train1}, set_{test1}>$.

G	Best T_G	Accuracy	FAR	FRR
1	1	73.95%	21.12%	30.99%
2	1	76.80%	34.66%	11.75%
5	3	83.00%	13.50%	20.50%
10	5	87.42%	14.40%	10.77%
20	10	89.89%	10.76%	9.45%
50	25	92.12%	8.19%	7.56%
100	49	93.37%	7.88%	5.37%
200	97	94.67%	7.57%	3.09%
500	252	96.54%	5.84%	1.08%
1000	514	97.68%	4.64%	0.00%

(b) Performance on validation dataset $<set_{train2}, set_{test2}>$.

G	T_G	Accuracy	FAR	FRR
1	1	74.47%	21.78%	29.27%
2	1	77.09%	35.83%	9.99%
5	3	84.15%	13.85%	17.84%
10	5	88.74%	14.89%	7.63%
20	10	91.89%	11.07%	5.16%
50	25	94.03%	8.54%	3.40%
100	49	94.85%	8.29%	2.01%
200	97	95.31%	7.91%	1.46%
500	252	96.15%	6.20%	1.50%
1000	514	97.44%	5.11%	0.00%

, where TP, FN, TN and FP stand for "true positive", "false negative", "true negative" and "false positive", respectively. These three metrics together comprehensively measure the performance of classifiers and have been widely used in various applications, such as human activity recognition and speaker identification.

D. Authentication Performance

As mentioned, our prototype system trains a one-class SVM classifier for each key, and combines the decisions of all the classifiers to make the final decision. In what follows, we analyze the dataset that we have collected to evaluate the authentication performance of our prototype system.

In the testing phase, we group the feature vectors in the testing dataset with size G. Technically, from the beginning of the dataset, we count G continues feature vectors, which are temporal adjacent, as a group. Each two adjacent groups have $G-1$ overlapping feature vectors. This overlapping strategy is similar to that adopted in [9].

For a group of feature vectors, our prototype predicts a feature vector as either positive (i.e., belonging to the owner) or negative (i.e., belonging to a non-owner) with the classifier that belongs to the feature vector's $keycode$. The final prediction on the whole group is made as follows. If the number of positive predictions in the group is greater than or equal to a pre-defined value T_G, the group is predicted as positive. Otherwise, it is predicted as negative.

A smaller T_G may result in higher false acceptance rate, while a larger T_G may result in higher false rejection rate. We determine the best T_G for a group size G by traversing all possible values of T_G (i.e., from 1 to G) in the training phase with data $<set_{test1}>$ and select the one that achieves the best tradeoff between FAR and FRR. Then we apply the best T_G of each size G on the testing data in the validation data set, i.e., $<set_{test2}>$.

Table I demonstrates the results. The left table shows the performance with the best T_G of each G on the training data set while the right table shows the performance with the best T_G of each G on the validation data set. The results in both tables are quite similar, which indicates the trained classifiers do not suffer from the overfitting problem since we follow the cross validation routine. Moreover, we observe that $accuracy$ keeps increasing with the increase of the group size G, and reaches around 90% when $G = 20$. It is because that a larger group size can smooth out noise and achieve better performance. This is an encouraging result which shows that SafeKey can authenticate a user who taps only 20 keys on the software keyboard with an accuracy of about 90%.

TABLE II
ACCURACY OF KEYS

Key	Accuracy	Training size	Testing size
o	77.73%	2541	1252
i	77.59%	3773	1921
a	77.27%	3749	1893
h	77.11%	3074	1541
e	77.10%	5799	2862
n	76.91%	3136	1541
[period]	75.93%	2784	1389
t	75.76%	2675	1325
[delete]	75.11%	3604	1867
[shift]	74.90%	3912	1959

We also analyze the separate accuracy achieved by each key's classifier. Table II lists the keys with the top ten accuracies. From the table, we observe that the accuracies are quite similar no matter where and how large the key is. Considering Table I, the final accuracy is improved with the combinatorial decision strategy adopted in our prototype system.

E. A Comparison with Two-Class Model

As mentioned, SafeKey takes advantage of one-class classifiers which require only the owner's data in the training phase. However, two-class classifiers have been widely used in existing works. They always have better performance than the corresponding one-class classifiers, but suffer from the fact that they need both positive and negative data in the training phase. Therefore, we conduct experiments to evaluate the authentication performance of the two-class model which is in contrast to the SafeKey's one-class model.

The experiments are almost same as those in Section III-D. The only difference is that when training a classifier, we not only use the owner's data, but also use non-owners' data. In

TABLE III
AUTHENTICATION PERFORMANCE, TWO-CLASS MODEL

(a) Performance on training dataset $<set_{train1}, set_{test1}>$.

G	Best T_G	Accuracy	FAR	FRR
1	1	81.64%	18.03%	18.69%
2	1	82.57%	29.61%	5.24%
5	3	89.95%	11.29%	8.81%
10	6	92.28%	6.93%	8.51%
20	11	94.07%	6.65%	5.21%
50	28	95.36%	4.69%	4.59%
100	55	96.08%	4.45%	3.38%
200	105	97.05%	4.91%	1.00%
500	285	98.10%	2.56%	1.23%
1000	602	99.24%	1.52%	0.00%

(b) Performance on validation dataset $<set_{train2}, set_{test2}>$.

G	T_G	Accuracy	FAR	FRR
1	1	81.98%	18.73%	17.32%
2	1	82.11%	30.86%	4.91%
5	3	90.16%	11.61%	8.07%
10	6	92.80%	6.92%	7.47%
20	11	94.29%	6.57%	4.85%
50	28	95.32%	4.51%	4.85%
100	55	95.97%	4.29%	3.76%
200	105	96.89%	4.82%	1.40%
500	285	96.91%	2.88%	3.29%
1000	602	95.98%	1.99%	6.06%

the training and testing phases, to obtain unbiased classifiers and performance results, we balance the number of the owner's feature vectors and the number of non-owners' feature vectors. Additionally, T_G of each size G is determined and applied in the same way as described in the previous subsection.

Table III demonstrates the results of the experiments. We can observe that for single key-tap prediction ($G = 1$), the two-class model achieves 82% accuracy, which is higher than 74% achieved by the one-class model. However, the difference of accuracy between the two models keeps decreasing with the growth of G. When $G = 20$, the two-class model's accuracy is 94% while the one-class model's accuracy is 92%.

We also observe that the two-class model has better FAR than the one-class model. It is because that the training algorithm of two-class SVM encodes non-owners' information into the classifiers. FAR sometimes is more important than FRR because it implies failures in protecting the mobile device while FRR only implies inconvenience to the owner. Nevertheless, when G becomes large enough, the one-class model can also achieve acceptable FAR. What is more, as elaborated before, one-class model is much more practical than two-class model in building user (re-)authentication systems.

IV. CONCLUSION

In this paper, we present SafeKey, a secure system for user re-authentication on mobile devices. SafeKey has been demonstrated to have the following features: (1) *Self-constructiveness*: SafeKey treats the user authentication problem as an one-class problem. It only needs the data collected from the owner to construct the classifier for user authentication. By doing this, it removes the assumption most existing works have made that the data from non-owners is available in the training phase. (2) *Adaptiveness*: SafeKey can automatically detect the changes of user input patterns on the on-screen keyboard and re-train its classifiers to adapt to these changes. (3) *Effectiveness*: SafeKey achieves about 90% accuracy for user text input with only 20 key-taps. (4) *Efficiency*: SafeKey's re-authentication algorithm executes fast in runtime and consumes limited computational and storage resources. (5) *Non-intrusiveness*: SafeKey automatically functions in background without any user intervention.

REFERENCES

[1] Cisco, "Cisco visual networking index: Global mobile data traffic forecast update, 2015–2020 white paper," http://www.cisco.com/c/en/us/solutions/collateral/service-provider/visual-networking-index-vni/mobile-white-paper-c11-520862.html, 2016.

[2] Symantec, "Norton survey reveals one in three experience cell phone loss, theft," https://www.symantec.com/en/au/about/newsroom/press-releases/2011/symantec_0208_01, 2011.

[3] Google, "Remotely ring, lock, or erase a lost device," https://support.google.com/a/answer/173390?hl=en, 2016.

[4] M. Frank, R. Biedert, E. Ma, I. Martinovic, and D. Song, "Touchalytics: On the applicability of touchscreen input as a behavioral biometric for continuous authentication," *IEEE Transactions on Information Forensics and Security*, vol. 8, no. 1, pp. 136–148, 2013.

[5] A. De Luca, A. Hang, F. Brudy, C. Lindner, and H. Hussmann, "Touch me once and i know it's you!: Implicit authentication based on touch screen patterns," in *Proceedings of CHI*, 2012.

[6] N. Sae-Bae, K. Ahmed, K. Isbister, and N. Memon, "Biometric-rich gestures: A novel approach to authentication on multi-touch devices," in *Proceedings of CHI*, 2012.

[7] L. Cai and H. Chen, "Touchlogger: Inferring keystrokes on touch screen from smartphone motion," in *Proceedings of HotSec*, 2011.

[8] E. Miluzzo, A. Varshavsky, S. Balakrishnan, and R. R. Choudhury, "Tapprints: Your finger taps have fingerprints," in *Proceedings of MobiSys*, 2012.

[9] L. Li, X. Zhao, and G. Xue, "Unobservable re-authentication for smartphones," in *Proceedings of NDSS*, 2013.

[10] N. Zheng, K. Bai, H. Huang, and H. Wang, "You are how you touch: User verification on smartphones via tapping behaviors," in *Proceedings of ICNP*, 2014.

[11] R. Raghavendra, C. Busch, and B. Yang, "Scaling-robust fingerprint verification with smartphone camera in real-life scenarios," in *Proceedings of BTAS*, 2013.

[12] S. Bazrafkan, T. Nedelcu, C. Costache, and P. Corcoran, "Finger vein biometric: Smartphone footprint prototype with vein map extraction using computational imaging techniques," in *Proceedings of ICCE*, 2016.

[13] K. B. Raja, R. Raghavendra, V. K. Vemuri, and C. Busch, "Smartphone based visible iris recognition using deep sparse filtering," *Pattern Recognition Letters*, vol. 57, no. C, pp. 33–42, 2015.

[14] V. Hodge and J. Austin, "A survey of outlier detection methodologies," *Artificial Intelligence Review*, vol. 22, no. 2, pp. 85–126, 2004.

[15] C.-C. Chang and C.-J. Lin, "Libsvm: A library for support vector machines," *ACM Transactions on Intelligent Systems and Technology*, vol. 2, no. 3, pp. 27:1–27:27, 2011.