Inferring Touch From Motion in Real World Data

Pascal Bissig, Philipp Brandes, Jonas Passerini, and Roger Wattenhofer

ETH Zurich, Switzerland,
firstname.lastname@ethz.ch,
http://www.disco.ethz.ch/

Abstract. Most modern smartphones are equipped with motion sensors to measure the movement and orientation of the device. On Android and iOS, accessing the motion sensors does not require any special permissions. On the other hand, touch input is only available to the application currently in the foreground because it may reveal sensitive information such as passwords. In this paper, we present a side channel attack on touch input by analyzing motion sensor readings. Our data set contains more than a million gestures from 1'493 users with 615 distinct device models. To infer touch from motion inputs, we use a classifier based on the Dynamic Time Warping algorithm. The evaluation shows that our method performs significantly better than random guessing in real world usage scenarios.

Keywords: motion sensing, side-channel attack, touch input

1 Introduction

Smartphones have become an integral part of our daily lives. Motion sensors measure the movement and orientation of such devices which are useful to, e.g., adjust the screen orientation or to control games. These sensors are not considered to reveal sensitive information. Therefore, they can be accessed by any application installed on the device, even by applications running in the background. This holds for Android as well as iOS devices, together making up more than 90% of the market share in mobile operating systems in 2014. For security reasons, the same does not apply for touch screen input. Only foreground apps are granted access to touch input data since it reveals, among other things, characters being typed on the on-screen keyboard which may include passwords or private information in text messages. However, when touching the screen of a phone, the phone will also move. Depending on the usage scenario this motion might be very small and hard to track, for example when the phone is lying on a table. Interaction with handheld mobile devices usually causes the device to move so much that the built in accelerometer and gyroscope sensors are able to track this motion. In addition to that, it has been shown that this motion varies, as different areas of the screen are being touched. This tight relation between touch and motion raises the question: how well one can infer touch from

motion input? Being able to do so presents a security threat for most of today's smartphones.

Mobile devices are by design used in a large variety of environments that directly influence the motion sensor readings. This drastically complicates the task of inferring touch from motion and therefore has to be taken into account when evaluating any inference mechanism. To build a data set that reflects variable real world environments, we collect data through an Android game. Players are not instructed to hold or interact with the device in a specific way. Therefore, we do not know or take into account if a player is sitting in a train or walking while playing.

The player's task is to memorize and imitate patterns that mimic the lock-screen patterns found on most current mobile operating systems.

We describe how an attacker could collect both touch and motion data to train a classifier, that can be used to derive touch input in any application being used on the device under attack. We use the Dynamic Time Warping (DTW) algorithm to compare and classify gestures and evaluate classification accuracy for touch inputs. By performing this side channel attack, an attacker can steal passwords or at least reduce the number of guesses required to do so. Our results are based on a large scale user study that covers arbitrary and unknown usage scenarios, 1'493 users and 615 of different Android device models. We show that varying environmental influences impact performance heavily.

2 Related Work

Touch-Motion Side Channel Attacks Motion sensors have previously been used for side channel attacks. In most cases data was recorded in controlled environments, thereby reducing the impact of real world effects such as varying user activities. An attacker cannot be sure that motion data was recorded when the user was sitting still, instead the user might be walking or riding a train.

Examples of such results include the paper by Cai and Chen [3], who showed that side channel attacks on touch input using motion sensors are feasible in a lab environment. Aviv et. al. [2] collected data from 26 participants while sitting or walking. Although this study helps understanding the effects of added disturbances, the environment is still controlled and known. The study with the largest data set has been performed by Cai et. al. [4]. They collected 47'814 keystrokes from 21 test persons, with 4 devices in a lab setting. Miluzzo et. al. [11] performed a study comparing multiple classification algorithms with data from 10 test persons while sitting or standing. Although their results identify how to best infer touch from motion input, their data set also limits the applicability of their results in the real world. The work of Xu et. al. [15] focuses on tap gestures and describes a game which collects training gestures when running in the foreground and test gestures when running in the background. However, their data set spans only three users and was recorded in a controlled environment. Owusu et. al. [13] focus on random forests to detect passwords on smartphones by analyzing the accelerometer readings of 4 test persons. Other studies [16,7] show

that smartphone users can be distinguished based on their touch and motion input behavior. Hinckley et al. [6] show that the combination of touch and motion data can be used to create novel ways to interact with our mobile devices.

In contrast to the papers described above, we focus on collecting data in uncontrolled environments. Namely, our data set originates from users that were primarily concerned with playing a game on their smartphone and were not instructed about how and where to do so. The environments may range from office spaces to airplanes or trains. Since our data acquisition process can be replicated by an attacker, our data set allows us to assess how realistic a side channel attack on touch input really is. We collected data from 1'493 test users, which generated more than a million gestures on 615 distinct device models. This is roughly 50 to 200 times more participants and up to 20 times more gestures than in previous studies. To our knowledge, there is no similar work with the same magnitude of collected data in similarly unconstrained environments.

Smartphone Side Channel Attacks Touch input was not the only target of side channel attacks using motion sensor data. Liu et. al. [8] tried to infer three-dimensional, free-hand movements using accelerometer readings. They collected 4'480 gestures from 8 test persons over several weeks. With (sp)iPhone, Marquardt et. al. [9] developed a mechanism to infer input on a physical keyboard by analyzing the motion sensor readings of a phone laying next to the keyboard. The proof of concept Android application Gyrophone [10] demonstrates that it is possible to recognize speech using a gyroscope sensor. Niu et. al. [12] used Dynamic Time Warping to measure the similarity of gestures to authenticate users. Recognizing ten distinct gestures, Chong et. al. [5] used the motion sensors to unlock the phone by performing a user defined series of gestures. Since the gesture detection is performed independently of the user and the raw information is not used in the authentication process, this is very similar to a password composed of ten letters that are entered through performing gestures instead of pressing keys.

3 Data Collection

To simulate a realistic attack scenario, we decided not to invite test persons into a test laboratory with a predefined and controlled environment. Instead, we developed an Android game, which we distributed on the Google Play store[1] to collect data. The same method can be used by an attacker and might already be exploited. In the game, the player's task is to memorize and reproduce patterns on the screen as shown in Figure 1. The patterns are displayed in the bottom half of the screen, where one usually finds the keyboard or pin input field. As the levels get harder, the grid resolution is increased from 3×3 to 4×4 touch elements, which we call cells. The game not only asks the user to press specific

[1] Game on Google Play. `https://play.google.com/store/apps/details?id=ch.ethz.pajonas.ba.imitationgame.android` (2015-03-13)

cells in this area, which we call tap gesture, but also to connect cells using swipe gestures. In contrast to an attacker, we informed users upon installation and first launch of the game that motion data is collected for research purposes and only when the game is running in the foreground. Touch input is measured in terms of x,y coordinates or a series of them for swipe gestures. In addition to touch input, the game also records the x,y,z coordinates of both accelerometer and gyroscope sensors built into the device. All recorded data is linked to a randomly generated unique id that is generated when the game is first installed. Since different users might play the game on the device, this unique id is device (and not user) specific. In addition to this, we collect basic device information such as the device manufacturer and type. When connected to a WLAN, the compressed log files are sent to a central database. When analyzing users with bad classification performance, we observed that their motion sensor measurements contain random readings close to zero. This indicates, that the device might not be moving enough, possibly being placed on a table or otherwise fixed. We excluded such users from our experiments. The number of users evaluated for each experiment are mentioned in the respective sections.

Fig. 1: Screens of the game showing the main menu, and both the 3×3 as well as the 4×4 grid the player interacts with during the game.

4 Preprocessing

The motion data is first segmented using the touch input as ground truth. To account for device motion that occurs before and after the screen is touched, we leave a variable amount of sensor readings before the touch gesture starts

and after it ends. The time window is at most 100 ms. Figure 2 illustrates the segmentation process in more detail. This segmentation can also be performed in an attack scenario, at least for the training data collection phase, using the same technique. For the classification task, we can use device events to segment motion data. For example, power on and unlock events provided by the operating system can be used to segment pattern or pin inputs performed to unlock the device. To remove the effects of gravity and gyroscope drift, we remove the mean values for each individual sensor axis. We thereby implicitly assume that the device orientation and velocity is the same at the beginning and at the end of each gesture.

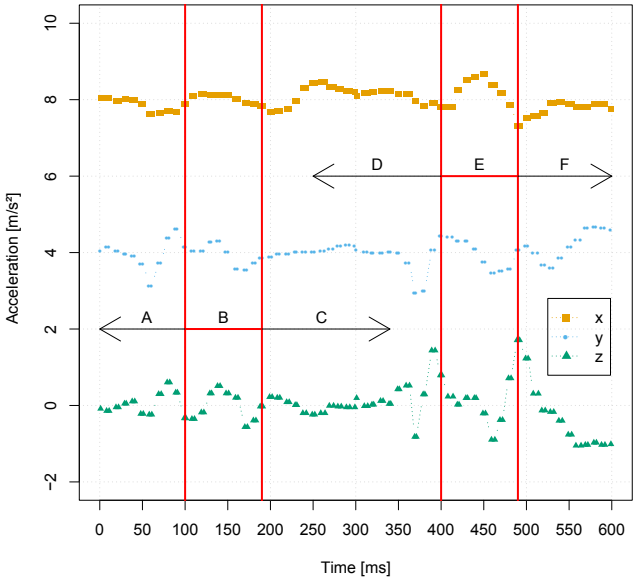


Fig. 2: Accelerometer readings containing two gestures before they get segmented into separated gestures. The vertical lines represent the touch down and touch up events respectively. Segments A, B, and C describe the first, D, E, and F the second gesture. A marks the pre-measurements, B encloses all measurements between the touch down and touch up events, and C contains the post measurements (the same applies to D, E, and F for the second gesture). C and D may overlap, but not C with E nor D with B. The length of the segments A and C (D and F respectively) may vary and are set to 150 ms in this example.

5 Classification

We do not attempt to guess the exact pixel the user touched, but rather predict the cell, which was pressed or swiped by the user. We build a classifier for

each separate unique id and assume an id to relate to one single player although multiple users might play the game on the same device. To quantify the similarity of two gestures, we use the DTW algorithm [1] with varying cost functions to compare individual time samples as described in Section 5.1. The user model M consists of 10 motion sensor recordings for each cell i on the grid M_i. In order to classify a test gesture t, our algorithm computes the DTW distance to all samples in M and selects the cell with the minimal DTW distance (see Equation 1). We also evaluated different metrics, such as the average and median DTW distance (Equation 2).

$$\text{class}(t) = \arg\min_{x} \min_{i \in M_x} \text{dtw}(t, i) \tag{1}$$

$$\text{class}(t) = \arg\min_{x} \left(\frac{1}{|M_x|} \sum_{i \in M_x} \text{dtw}(t, i) \right) \tag{2}$$

However, the average DTW distance is not robust against outliers in the training set, and therefore was expected to produce worse results than the min and median distances. Alternative classification techniques used in similar work are Hidden Markov Models [2] or feature based approaches, Support Vector Machines [14].

5.1 Dynamic Time Warping Cost Functions

To perform the time series analysis using the DTW algorithm, we need to select features, which we can use to describe the distance or matching cost between two sensor events from two different gestures. Each motion sensor event consists of 3 accelerometer and 3 gyroscope measurements, one for each spatial dimension. These 6 coordinates form a feature vector $A = \{\text{acc}_x, \text{acc}_y, \text{acc}_z, \text{gyro}_x, \text{gyro}_y, \text{gyro}_z\}$, representing the values of the x, y, and z accelerometer and gyroscope coordinates of a single sensor event in a touch or swipe gesture. One possible metric to calculate the distance between two sensor events is the Euclidean distance, as Niu et al. suggest in their work [12]. Cai et al. [4] propose a different feature, calculating the two-argument arctangent (atan2) using the x and y axis of the accelerometer readings, arguing that motion data on the z axis is not a good feature to infer keystrokes (see Equation 3). We designed a new metric which pairwise calculates atan2 for all accelerometer and gyroscope axes combinations and then sums up their absolute differences (Equation 4). The equations below show how those three metrics are used to compare two sensor events i, and j in two different gestures A and B. Variations of the metrics above and others like the Manhattan distance or the L_∞-norm are also included in our experiments. The Manhattan distance is the sum of the accelerometer and gyroscope differences between two measurements.

$$c_{acc}^{xy}(A_i, B_j) = \left| \text{atan2}(A_{acc}^{i,x}, A_{acc}^{i,y}) - \text{atan2}(B_{acc}^{j,x}, B_{acc}^{j,y}) \right| \tag{3}$$

$$c_{acc}(A_i, B_j) = c_{acc}^{xy}(A_i, B) + c_{acc}^{xz}(A_i, B_j) + c_{acc}^{yz}(A_i, B_j)$$

$$c_{sum}(A_i, B_j) = c_{acc}(A_i, B_j) + c_{gyro}(A_i, B_j) \tag{4}$$

5.2 Dynamic Time Warping Penalty

In order to penalize sequences whose time axis needs to be stretched a lot, we employ different penalization factors p. See Equation 5 for the recursive definition of an entry in the DTW matrix $d(i, j)$. In the penalization experiment, we use the c_{sum} distance metric to compare two measurements A_i and B_j at times i and j respectively. By increasing p, the cost of advancing time i and j unevenly is penalized. This leads to a higher DTW matching cost for sequences of uneven length or speed.

$$\begin{aligned} d(i,j) = \min\{\ & d(i-1, j-1) + c_{sum}(A_i, B_j) \\ & d(i, j-1) \quad\ + c_{sum}(A_i, B_j) \cdot p \\ & d(i-1, j) \quad\ + c_{sum}(A_i, B_j) \cdot p\ \} \end{aligned} \tag{5}$$

6 Data Set

At the end of the 4 month data collecting phase, the game reached 2'049 installations. Most of the users are from India, USA, Italy, or Switzerland. The Android application received 70 ratings in the Google Play Store, with an average rating of 4.01 stars out of 5. The most used device to play the game is the Google Nexus 5. Figure 3 shows that most players' phones are produced by Samsung, and the most popular Android version is 4.4. Data has been collected from 615 device models with over 38 different screen sizes. The server application received data

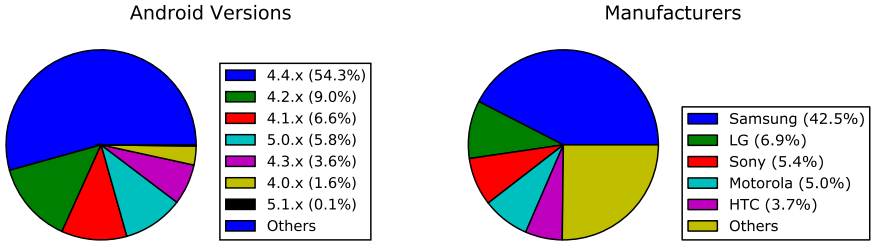


Fig. 3: Variety of the devices and Android versions on which we collected our data set. Data has been collected from 615 distinct device models from 25 manufacturers with over 38 different screen sizes.

from 1'493 users, who played a total of 87'962 levels. With an average of 15 gestures per level, this corresponds to more than a million collected gestures. Since the data has not been collected in a laboratory, it contains unknown external influences on the motion data, devices with malfunctioning motion sensors, and users with very few gestures.

7 Evaluation Setup

A training gesture is a gesture for which we know the motion sensor readings as well as the ground truth touch data, which we collected with our Android application. A test gesture on the other hand, is a gesture for which the touch input is ignored and the task is to infer the correct touch gesture using only the motion sensor readings. Only users with working accelerometer and gyroscope sensors are included in the experiments. Not all players generated enough data to generate a model for each cell. In these cases, we only classify cells with at least 10 recorded training gestures. Hence, a user might have fewer than 9 or 16 cells with enough training data. The random guessing probability is adapted to compensate for the reduced solution space of the classification problem. To generate enough training data for every single cell, a user is required to interact with the Android application for about 30 minutes. Depending on the experiment, between 10 and 500 players were used to evaluate performance changes.

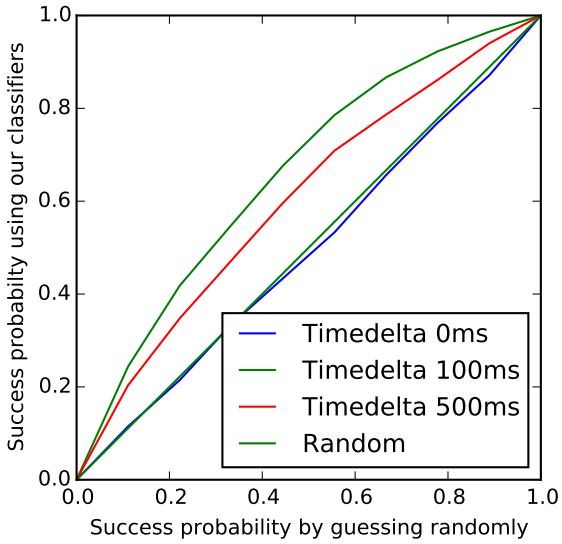


Fig. 4: Comparison of different segmentation time windows. A larger window means that there are more pre- and post- measurements. In this plot, tap gestures on the 3×3 board from 14 users have been evaluated

7.1 Segmentation Time Window

The segmentation time window controls the number of pre- and post-measurements of each gesture. This window can be varied to optimize the classification perfor-

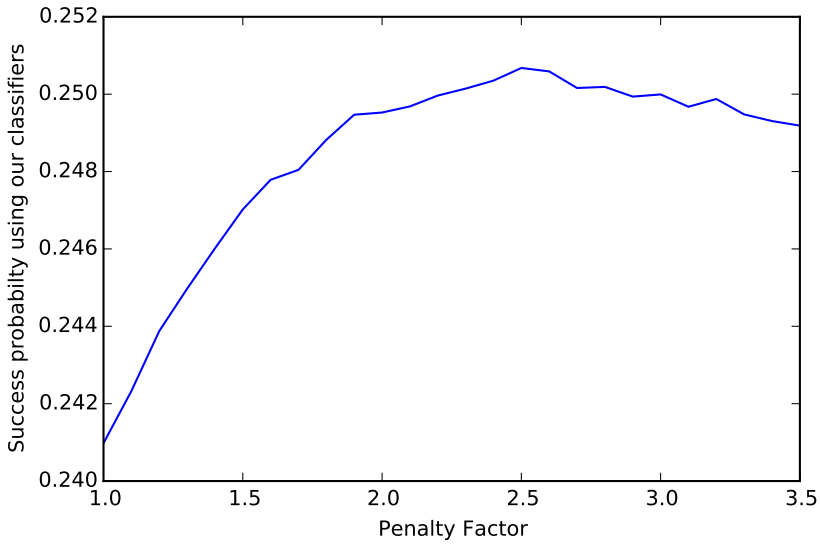


Fig. 5: Influence of different penalties on the classification performance. In this experiment, all users have been evaluated for tap gestures on the 3×3 board. We have evaluated the probability of correctly guessing the label with the first attempt.

mance since the device moves already before touch input is registered. For this experiment, we use our sum of atan2 cost function as described in Equation 4. The DTW penalty is set to 240% and we use the min classification method as described in Equation 1. Figure 4 shows different time windows of the length 0 ms, 100 ms, and 500 ms. Including no measurements that are recorded before the touch gesture starts produced bad results. This leads to the conclusion that the motion of the device before the touch event actually starts is the most discriminant. The time window of 500ms also performs significantly worse than 100ms, which is why we chose 100ms for all the other experiments. In our work, we do not focus on the segmentation of the test gestures. To detect the unlock pattern, it seems to be sufficient to trigger the measurements using the unlock events of the smartphone. To infer PIN entries or multiple gestures, one would need to segment the test gestures. According to previous results [4,15], there exist promising segmentation methods to achieve this.

7.2 DTW Penalty

To evaluate the effect of varying DTW time penalties, we analyzed the classification results from all players on the 3×3 grid utilizing the sum of atan2 metric as described in Equation 4 and the min classification method as described in

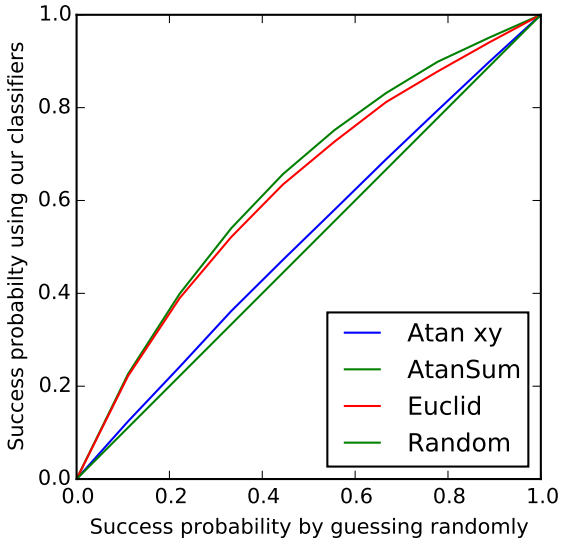


Fig. 6: Comparison of the distance metrics used in the DTW algorithm (described in Section 5.1). The sum over all atan2 features performs best and is used as the distance metric for the other experiments. In this plot, all users have been evaluated for tap gestures on the 3×3 board.

Equation 1. We evaluate the penalty in the range of $p = 100\%$ to $p = 350\%$ in 10% increments. Figure 5 shows the effect of a varying DTW penalty. As expected, penalties that are very small or large result in worse performance. For very small penalties, sequences can be stretched beyond what is to expect due to the natural variance each user causes. In case of very high penalties, the DTW algorithm mostly matches sequences without stretching either time axis, resulting in a score that is very close to the sum of all corresponding sample costs. The best classification results can be achieved with p at 240% but as one can see, the performance differences are marginal for non extreme choices of p.

7.3 *DTW* Cost Functions

Figure 6 shows the comparison of the four presented distance metrics in the DTW algorithm. For this experiment, all users were evaluated using a DTW penalty of 240%. Our proposed sum over all atan2 features metric performs slightly better than the atan2 function and is therefore chosen as the distance metric in all other experiments. The euclidean distance function performs worse than both of the other distance metrics, but still much better than other metrics we tested (Manhattan, L_∞) which we omit on this plot.

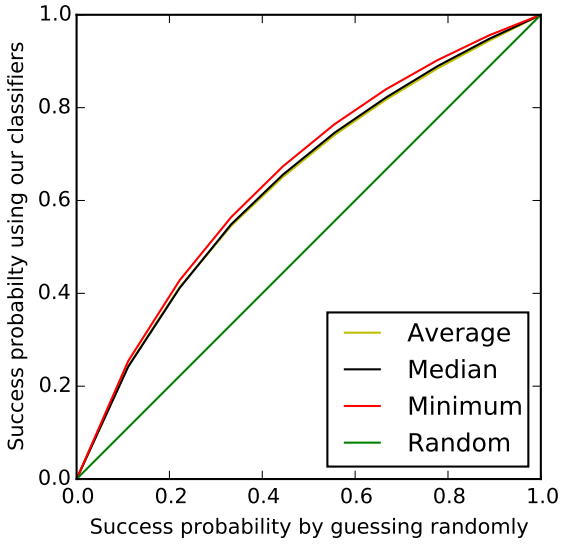


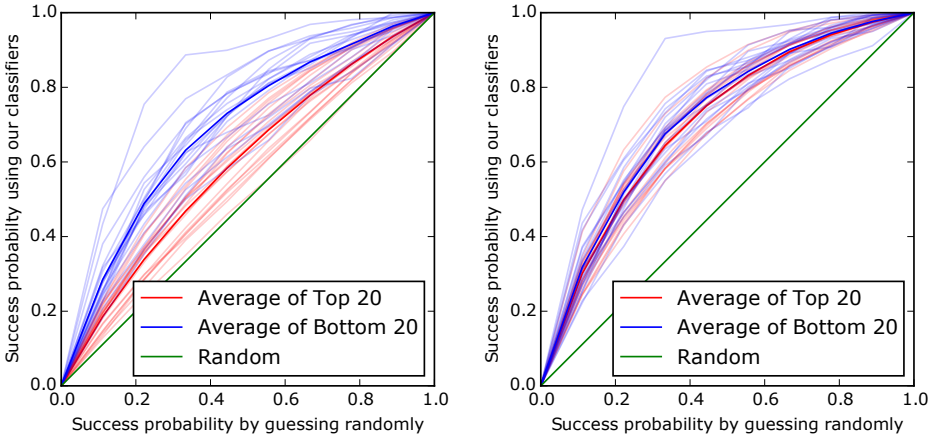
Fig. 7: Comparison of the classification methods min, mean and median as described in Section 5. The min metric performs best and is used for the other experiments. In this plot, all users have been evaluated on the 3×3 board.

7.4 Classification Methods

The performance comparison for the presented classification distance metrics for all users on the 3×3 grid is shown in Figure 7. The DTW penalty was set to 240%. Interestingly, performance variations are insignificant for all three methods. Since our training sets may include outliers, we expected the average method to perform significantly worse than the min and median method, respectively. If the user e.g., bumped into someone or walked up steps while performing the tap gesture, then this outlier will skew the results and make the classification task more difficult. The minimum distance method performed best in this experiment and is therefore used in the other evaluation tasks.

7.5 External Influences

To evaluate the impact of changing environments, we compare the performance of heavy- and light-users while using a fixed training set of 10, a DTW penalty of 240%, the min classification method (Equation 1), and the sum of atan2 features. Heavy users played more levels and hence, produced more data. We expect their data set not only to be bigger, but also to contain more varied environments. Since we limit the training set size to 10 for both user groups, we expect the performance for heavy users to be bad since the small training set cannot capture all environments the game was played in. Figure 8a shows the 20 users with the



(a) Performance comparison between heavy- and light users with fixed training set size. The environments in which heavy users played the game cannot be captured by a small training set size. Therefore, small training sets are sufficient, as long as the environmental effects are similar in both the training and testing phase.

(b) Impact of varying training set sizes on the classification performance. In this experiment, all training samples were used for each user on the 3×3 grid. Users with the most training data (in blue) show similar performance as compared to the users with the least training data (in red).

Fig. 8: Classification performance heavily depends on the range of environments in which the motion data is recorded.

most collected gestures (heavy) in blue and the 20 users with the least gestures in red (light) out of more than 200 users in total. Guessing the correct cell in the first try for the bottom 20 users is with 29% roughly 10% higher than for the top 20 users (19%).

7.6 Training Set Size

The results in the previous section beg to evaluate the same two user groups while using larger training sets for the heavy users. Performance for heavy users should increase as the training set captures more environments. To evaluate the effect of varying training set size, we lifted the restriction to only use 10 and instead trained our model with as all available samples for each user. This means that only the test gesture is excluded from the training set. Thereby we remove the advantage of light users being able to capture a larger fraction of the environmental effects in the training set when compared to the heavy users. In our data set this means that the training set can be up to two orders of magnitude larger. Classification was performed using the min metric (Equation 1). Figure 8b shows the 20 users with the most collected gestures in blue and the 20 users with the least gestures in red (out of more than 200 users in total). The classification

rate of the top 20 users differs insignificantly (1.5% on the first guess) from the bottom 20.

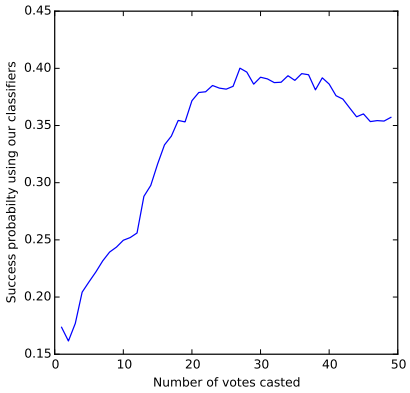
Since both heavy- and light-users are tested using training sets that capture all environments the game was played in, performance is consistent for both user groups. For an attacker, this means that small training set sizes are sufficient, as long as the environment under which touch input inference should be performed is similar to the one predominant during training data set collection.

Both the experiment on the training set size, as well as the one on the environmental effects were performed using the exact same two user groups. As long as the training set captured the external influences affecting the classification phase, performance is insensitive to the size of the training set. Collecting a large training set therefore helps capturing more environmental influences but does not allow the classification accuracy to improve significantly once an environment has been captured.

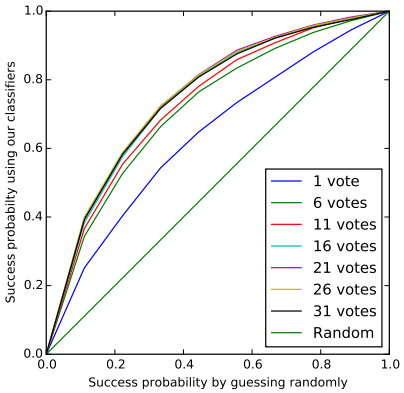
7.7 Repeated Attack

In this section we try to emulate an attack on the user's pin code to unlock the screen. Since users enter the same pin code over and over again, an attacker could use multiple motion measurements to improve the guessing accuracy. We know that after turning on the screen, the first number entered is always the first digit of the pin code. Thus, instead of using one test gesture, we use several of them. The task is now to classify these gestures, about which we know that they belong to the same label, but not to which one. The simplest approach is to cast a vote for each test gesture's most likely label according to the previously described method and then pick the label with the highest number of votes. Note that we limit our setting to k gestures in order to evaluate the influence of the number of votes on the classification accuracy we can achieve. If a user has more than k gestures, we limit it to k artificially and we keep the training set size fixed to 10.

The results are shown in Figure 9. As one can seen in Figure 9a, the chance to correctly guess the label in the first attempt increases with k. The chance dramatically increases for very small values of k. Performance stagnates around 40% when using more than 20 test gestures at once and starts deteriorating when using more than 35 test gestures. We believe that this is because of the limited training set size for users as discussed in Section 7.6. If we only consider users with more than 30 gestures per label, then these users need to have a lot of gestures and thus have played the game in varying environments. Hence, the more we increase k, the noisier the data gets. Thus, the advantage of having more votes to cancel out noise is balanced out by more noise introduced by later samples. As show in Figure 9b the performance not only increases for the first guess, but helps predicting the correct gesture with higher accuracy also for the following guesses. In our case, the peak performance of 40% of first guesses being correct was reached when using 27 test gestures. This means that an attacker can more easily guess pins or passwords that are repeatedly entered.



(a) The probability of guessing the correct label for a given set of test gestures improves from less than 20% to more than 40% when increasing k from 1 to 27 test gestures



(b) Not only the first guess accuracy improves, but all consecutive guesses are more accurate as well. More than 50% of gestures are classified correctly in two guesses.

Fig. 9: Credentials are often entered repeatedly. By collecting the data repeatedly the same input, we can reduce the influence of noise in the measurements. The accuracy of the attack increases as the number test gestures available to guess one label grows.

In addition to that, the attacker needs to solve the problem of recognizing repeated inputs. In case of device unlock pins or patterns, this is easily achieved through events triggered by the operating system.

8 Additional Observations

We observed a heavy preference in which direction a gesture is performed. Most people have a preference on how they imitate a certain pattern. A pattern with a straight horizontal line can either be drawn with a swipe gesture from left to right or a swipe gesture from right to left. The same applies for vertical or diagonal lines. We analyzed the behavior of 452 users. The results are shown in Figure 10. One can see that most of the people prefer to perform the vertical gestures downwards and the horizontal gestures from left to right. When it comes to the diagonal gestures, there is an overall preference for the "Down - Right" gesture instead of the opposite direction, but for the "Up - Right" respectively "Down - Left" gestures, there is less of a general preference.

The profiles created in the previous subsection may reveal further information about the user. For single test persons, we observed that the gesture preference may be related to the handedness of the user.We could not confirm this claim,

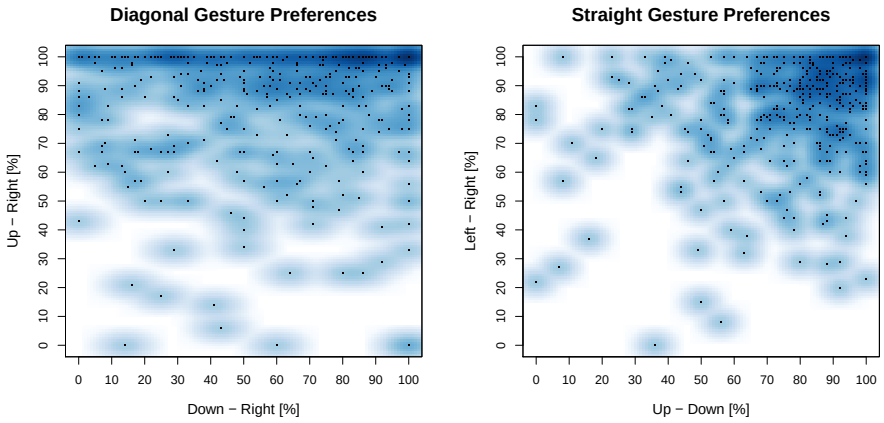


Fig. 10: Gesture preferences of 452 analyzed users. "Down - Right" indicates a downward diagonal line from left to right, "Up - Right" an upward diagonal line from left to right. "Up - Down" is a downward, vertical line, "Left - Right" a horizontal line from left to right.

since we did not collect the corresponding ground truth data with the Android application. In future work, one could collect this data from test users to answer this question.

9 Conclusion

In this paper, we discussed a side channel attack on touch input by analyzing motion sensor readings. Firstly, we collected data by distributing an Android application. Secondly, we trained a *DTW* based classifier using the collected data to infer touch gestures. In contrast to similar work, we collected real world data in a way an attacker could also do. The evaluation has shown that the side channel attack presents a realistic threat. Especially for touch input which is repeated often, such as unlock patterns or pin codes, motion sensor data can help an attacker to guess the correct touch input.

As opposed to software vulnerabilities, the side channel attack we analyzed in this paper is not caused and cannot be fixed by app developers. Background access to motion sensors needs to be limited on the operating system level because otherwise no application can protect itself against these attacks. Since we expect motion sensors to become more accurate in the future, the risk of a successful side channel attack grows even further. With mobile payment solutions becoming more and more popular, the incentive to spy on touch input on mobile devices increases.

References

1. Dynamic time warping. In *Information Retrieval for Music and Motion*, pages 69–84. Springer Berlin Heidelberg, 2007.
2. Adam J. Aviv, Benjamin Sapp, Matt Blaze, and Jonathan M. Smith. Practicality of accelerometer side channels on smartphones. In *ACSAC*, pages 41–50. ACM, 2012.
3. Liang Cai and Hao Chen. Touchlogger: Inferring keystrokes on touch screen from smartphone motion. In *HotSec*, pages 9–9, 2011.
4. Liang Cai and Hao Chen. On the practicality of motion based keystroke inference attack. In *TRUST*, volume 7344 of *Lecture Notes in Computer Science*, pages 273–290. Springer, 2012.
5. Ming Ki Chong, Gary Marsden, and Hans Gellersen. Gesturepin: using discrete gestures for associating mobile devices. In *Mobile HCI*, pages 261–264. ACM, 2010.
6. Ken Hinckley and Hyunyoung Song. Sensor synaesthesia: touch in motion, and motion in touch. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, pages 801–810. ACM, 2011.
7. Sarah Martina Kolly, Roger Wattenhofer, and Samuel Welten. A personal touch: Recognizing users based on touch screen behavior. In *Proceedings of the Third International Workshop on Sensing Applications on Mobile Phones*, page 1. ACM, 2012.
8. Jiayang Liu, Zhen Wang, Lin Zhong, Jehan Wickramasuriya, and Venu Vasudevan. uwave: Accelerometer-based personalized gesture recognition and its applications. In *PerCom*, pages 1–9. IEEE Computer Society, 2009.
9. Philip Marquardt, Arunabh Verma, Henry Carter, and Patrick Traynor. (sp)iphone: decoding vibrations from nearby keyboards using mobile phone accelerometers. In *ACM CCS*, pages 551–562. ACM, 2011.
10. Yan Michalevsky, Dan Boneh, and Gabi Nakibly. Gyrophone: Recognizing speech from gyroscope signals. In *23rd USENIX Security Symposium*, pages 1053–1067, San Diego, CA, August 2014. USENIX Association.
11. Emiliano Miluzzo, Alexander Varshavsky, Suhrid Balakrishnan, and Romit Roy Choudhury. Tapprints: your finger taps have fingerprints. In *MobiSys*, pages 323–336. ACM, 2012.
12. Yuan Niu and Hao Chen. Gesture authentication with touch input for mobile devices. In *MobiSec*, volume 94 of *Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering*, pages 13–24. Springer, 2011.
13. Emmanuel Owusu, Jun Han, Sauvik Das, Adrian Perrig, and Joy Zhang. Accessory: Password inference using accelerometers on smartphones. In *Proceedings of the Twelfth Workshop on Mobile Computing Systems and Applications*, HotMobile '12, pages 9:1–9:6, New York, NY, USA, 2012. ACM.
14. Jiahui Wu, Gang Pan, Daqing Zhang, Guande Qi, and Shijian Li. Gesture recognition with a 3-d accelerometer. In *UIC*, volume 5585 of *Lecture Notes in Computer Science*, pages 25–38. Springer, 2009.
15. Zhi Xu, Kun Bai, and Sencun Zhu. Taplogger: inferring user inputs on smartphone touchscreens using on-board motion sensors. In *WISEC*, pages 113–124. ACM, 2012.
16. Nan Zheng, Kun Bai, Hai Huang, and Haining Wang. You are how you touch: User verification on smartphones via tapping behaviors. In *Network Protocols (ICNP), 2014 IEEE 22nd International Conference on*, pages 221–232. IEEE, 2014.