Snoopy: Sniffing Your Smartwatch Passwords via Deep Sequence Learning

CHRIS XIAOXUAN LU, University of Oxford, UK
BOWEN DU and HONGKAI WEN*, University of Warwick, USA
SEN WANG, Heriot-Watt University, UK
ANDREW MARKHAM and IVAN MARTINOVIC, University of Oxford, UK
YIRAN SHEN, Harbin Engineering University, China
NIKI TRIGONI, University of Oxford, UK

Demand for smartwatches has taken off in recent years with new models which can run independently from smartphones and provide more useful features, becoming first-class mobile platforms. One can access online banking or even make payments on a smartwatch without a paired phone. This makes smartwatches more attractive and vulnerable to malicious attacks, which to date have been largely overlooked. In this paper, we demonstrate Snoopy, a password extraction and inference system which is able to accurately infer passwords entered on Android/Apple watches within 20 attempts, just by eavesdropping on motion sensors. Snoopy uses a uniform framework to extract the segments of motion data when passwords are entered, and uses novel deep neural networks to infer the actual passwords. We evaluate the proposed Snoopy system in the real-world with data from 362 participants and show that our system offers a ~ 3-fold improvement in the accuracy of inferring passwords compared to the state-of-the-art, without consuming excessive energy or computational resources. We also show that Snoopy is very resilient to user and device heterogeneity: it can be trained on crowd-sourced motion data (e.g. via Amazon Mechanical Turk), and then used to attack passwords from a new user, even if they are wearing a different model.

This paper shows that, in the wrong hands, Snoopy can potentially cause serious leaks of sensitive information. By raising awareness, we invite the community and manufacturers to revisit the risks of continuous motion sensing on smart wearable devices.

CCS Concepts: • **Security and privacy** → Authentication; • **Human-centered computing** → Mobile devices;

Additional Key Words and Phrases: Smartwatch, APL, Motion Sensors

ACM Reference Format:
Chris Xiaoxuan Lu, Bowen Du, Hongkai Wen, Sen Wang, Andrew Markham, Ivan Martinovic, Yiran Shen, and Niki Trigoni. 2017. Snoopy: Sniffing Your Smartwatch Passwords via Deep Sequence Learning. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.* 1, 4, Article 152 (December 2017), 29 pages. https://doi.org/10.1145/3161196

*Hongkai Wen is the corresponding author.

Authors' addresses: Chris Xiaoxuan Lu, Department of Computer Science, University of Oxford, Oxford, OX1 3QD, UK; Bowen Du; Hongkai Wen, University of Warwick, Department of Computer Science, Coventry, CV4 7AL, USA; Sen Wang, Department of Computer Science, Heriot-Watt University, Oxford, OX1 3QD, UK; Andrew Markham; Ivan Martinovic, Department of Computer Science, University of Oxford, Oxford, OX1 3QD, UK; Yiran Shen, College of Computer Science and Technology, Harbin Engineering University, Harbin, 150001, China; Niki Trigoni, Department of Computer Science, University of Oxford, Oxford, OX1 3QD, UK.

2474-9567/2017/12-ART152 $15.00
https://doi.org/10.1145/3161196

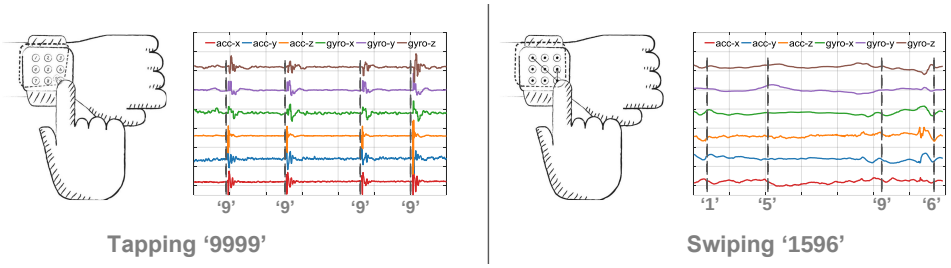

Fig. 1. An example of motion sensor data changes induced by swiping a pattern-lock on a smartwatch. Tapping or swiping passwords does not follow uniform motion and is very challenging to distinguish individual digits, let alone reveal the entire code.

1 INTRODUCTION

Smartwatches are becoming increasingly ubiquitous: it is expected that the global smartwatch market has a potential to reach $32.9 billion by 2020 [30]. They are now deeply embedded in our daily lives, and over time can accumulate a variety of sensitive and important information such as emails, contacts and payment details. Due to their current role as an extension to the smartphones, the security and privacy of smartwatches have been overlooked, and instead delegated to the paired phones. However, driven by the major players such as Google and Apple, smartwatches are becoming more independent and can act as first class citizens in the mobile ecosystem: they are no long just secondary displays, but are able to offer all basic functionalities without the presence of smartphones. For instance, it is already possible to pay via a smartwatch without even needing to carry a smartphone [2, 6, 11], and many recent apps on smartwatches such as fitness tracking, well-being monitoring, and messaging (email/text) apps can work independently of phone usage.

These increased functionalities make smartwatches more useful, but also attract malicious attacks which traditionally target smartphone class devices only. Smartwatches are typically secured using a 4 digit PIN or a pattern lock, e.g., Android Pattern Locks (APLs). These are used not only to unlock the phone, but also to authenticate the payments. In practice, the consequences of such an attack can be more serious than just security breach of the smartwatch screen lock [33]: as shown in our user study (discussed in Sec. 7), over 80% of 745 anonymous participants have a frequent habit of reusing the same passwords across services e.g. PayPal, card payments e.g. ATM PIN codes or even physical security e.g. home alarm systems. Therefore compromising a smartwatch password could lead to a series of cyber and physical attacks.

There has been a solid body of work on the similar problem of attacking passwords on smartphones, including analysing oily residues on the screen [4], video footage [43], radio signal perturbations [19] and motion sensor data [8]. In particular, attacking passwords by eavesdropping motion data is popular, since motion sensors are commonly sampled by a wide variety of applications, e.g. those designed for positioning, fitness tracking and activity recognition. In addition, giving access to on-board motion sensors appears to be innocuous, and many users (76% according to our user study) would grant instantly. Motion sensors leak information about the location of events such as taps through small changes in orientation and impacts. Existing techniques for cracking PINs on smartphones typically segment digits by extracting tap events and then use the extracted features to tell which digit has been pressed. As such, existing techniques rely on significantly handcrafted features and ad hoc approaches for digit segmentation.

Smartwatches with their smaller form factor compound the password classification problem, making it far more challenging than the smartphone case. Fig. 1 shows examples where the motion sensor data changes as the user swipes an APL on a smartwatch. As can be seen, the motion induced by password entries is small and

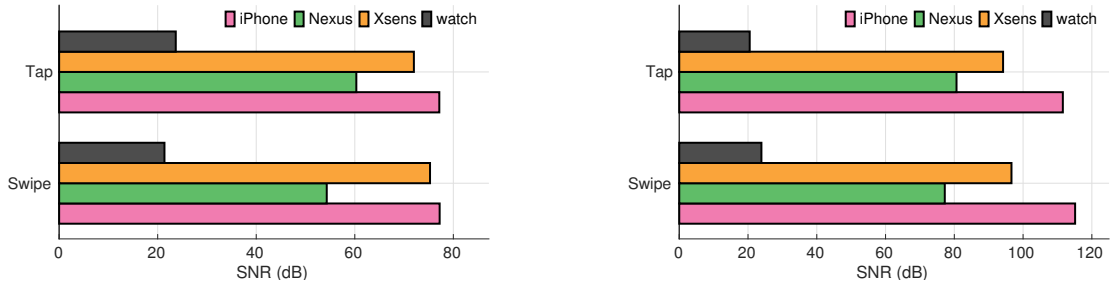

Fig. 2. Signal-to-noise ratio (SNR) of motion sensors on smartphones, high-end IMUs and smartwatches. Left: Accelerometers; Right: Gyroscopes

not easily segmented. Fig. 2 highlights this by considering the signal-to-noise (SNR) ratio of motion sensors on different devices. We see that motion signals on smartwatches are far noisier, and can be 20-40dB worse than that of smartphones or high-end IMUs. In the presence of such low SNR, existing techniques designed for smartphones [8] typically fail to work. This is due to the reliance on hand-engineered features, which are not robust to variability across users and devices, particularly given the much weaker and noisier motion signals on smartwatches.

Although there are a number of papers which look at cracking PINs on smartphones, to date, only one paper has considered the issue of revealing APLs. This is because APLs can have an arbitrary length and hence have significantly more possible combinations e.g. 60 passwords vs 10k for a 4 digit PIN. In this paper, we provide a *universal* data driven technique for inferring both APLs and PINs on smartwatches. This requires no handcrafted feature extraction or digit segmentation and is able to generalize well to the problem of arbitrary length APLs, even when faced with the extremely low SNRs found on smartwatches. Our novel deep learning approach, based on recurrent neural networks (RNNs), exhibits a 3-4 fold increase in accuracy compared with the state-of-the-art. We propose two different architectures. The first exploits the skewed distribution of passwords to perform entire passwords inference. This technique shows superior results on popular passwords. The second is capable of digit level inference i.e. it can generalize to any password that may or may not be present in a training database. Although we have only considered the challenges of smartwatches in this paper, this technique would easily work on the high SNR signals found on smartphones as well. In addition, we propose an adaptive motion sensing technique to detect the password in low sampling rate and it only sends motion data of the candidate password events to the server, minimizing battery and bandwidth usage to mimic a Trojan fitness app more readily. In summary, the contributions of this paper are:

- We present Snoopy, the first system that demonstrates the feasibility of intercepting password information entered on smartwatches by sensing resulting motion data.
- Snoopy is also the first approach that can infer universal APLs, a significantly more difficult problem than PINs, due to the challenges of digit segmentation.
- We propose a *universal* password inference mechanism based on deep recurrent neural networks that is able to attack PINs and APLs. We present two variants, one which cracks popular passwords and another which infers arbitrary passwords. Our system does not require any handcrafted features, only a crowdsourced training dataset.
- We have conducted a user study and collected over 1,000 answered questionnaires, which shows that the affected population of smartwatch users is nonnegligible and the majority of users are not aware of the potential password leak on smartwatches via motion data and its consequences.

- We have extensively evaluated the proposed Snoopy system, using data from over 360 distinct participants and > 60K password entries on both Android and Apple devices. Our results show that the Snoopy achieves a 3-4 fold improvement in correctly inferred passwords compared to competing techniques.

The remainder of the paper is organised as follows. Sec. 2 covers the technical background, while Sec. 3 presents the overview of the proposed Snoopy system. Sec. 4 proposes a uniform approach to extracting the relevant part of the motion signal that corresponds to entering PINs and APLs. Sec. 5 proposes two models of inferring the actual contents of passwords using deep neural networks. Sec. 6 evaluates the proposed extraction and inference approaches and compares them with competing state-of-the-art methods. Sec. 7 reports the results of our user study, while Sec. 8 discusses the energy/accuracy tradeoff of Snoopy and possible countermeasures. Finally, Sec. 9 presents an overview of related work, and Sec. 10 concludes the paper and points to directions for future work.

2 TECHNICAL BACKGROUND

2.1 Tapped vs. Swiped Passwords

There are two predominant types of password input mechanism on smartwatches (also on smartphones): tapped and swiped passwords [39].

For iOS platforms, the default password type is four digit PIN, where the users *tap* their passwords on the screen when prompted. A four digit PIN has 10, 000 possible combinations. It is possible to use longer passwords, but in this paper, we only consider the four digit PIN.

For the Android platform, users have the option to use a PIN or a graphical pattern lock (also known as APL), where the users *swipe* a pattern over a three-by-three matrix of dots (see Fig.1 for an example). Unlike the numerical passwords where the users can choose freely from ten possible digits at each tap, the smartwatch operating systems typically have certain constraints over the trajectories of the swiped patterns. For instance as shown in Fig.1, starting from the top left dot, it is only possible to swipe towards four reachable neighbours: the immediate right and the three dots in the second row. Therefore, the size of the search space for swiped passwords is restricted to a maximum of 389, 112 [4].

In practice, for both PINs and APLs, if one fails to input the correct password three times the smartwatch will prevent any further attempts for a few minutes. When the number of failed attempts reaches a threshold, typically ten, the smartwatch can enter 'lost' mode, e.g. erase all data.

2.2 Motion Induced by Password Input

Intuitively, entering both tapped and swiped passwords will induce forces and orientation changes on the smartwatch [37]. Since human skin has a certain level of elasticity, tapping on the smartwatch screen will cause minor displacement at the contact point along the vertical direction, i.e. the watch body will rotate for a small angle. Tapping causes an underdamped impulsive wave to develop, which causes small oscillations, shown in Fig. 1. On the other hand, when swiping passwords, the pressing and friction force between the user's finger and touch screen will "drag" the smartwatch to move along both vertical and horizontal directions. This gives rise to small slip-pulse waves which have a longer duration than impulsive taps, as shown in Fig. 1.

In practice, induced motion can be picked up by the Inertial Measurement Units (IMUs) embedded on most of the commercial smartwatches. IMU sensors have been widely used in many mobile sensing scenarios, since they are able to capture displacement and rotation of the devices in 3-D space, and become increasingly cheap and power efficient. Concretely in this work we consider both accelerometers and gyroscopes, which capture the linear acceleration and angular velocity (roll, pitch and yaw) with respect to the three axis. By default the IMU sensors on most smartwatches are set to be always-on, continuously sensing motion for various applications such as gesture recognition, localisation, and fitness monitoring. This can lead to involuntary information leakage, which may be leveraged by malicious parties to infer private and valuable data such as passwords.

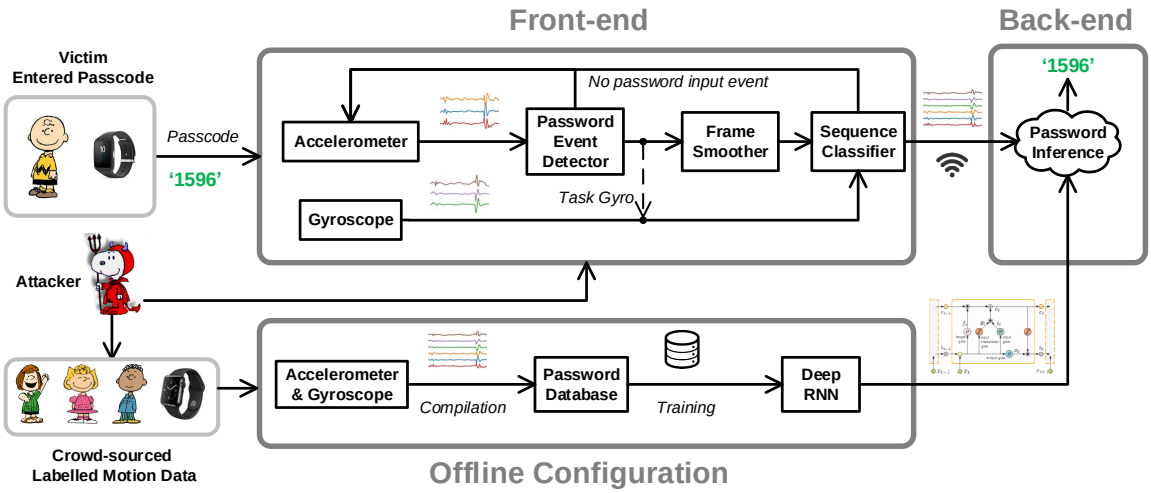

Fig. 3. System overview:The attacker builds a deep RNN classifier using crowd-sourced data. On a victim's smartwatch, a trojan app uses an adaptive sampling scheme to record and identify a victim's motion data. Candidate password sequences are uploaded to the server. The back-end server runs the trained deep RNN classifier to infer possible passwords. Note, training is only required by the attacker; no training is needed by the victim.

3 SYSTEM OVERVIEW

3.1 Attack Assumptions

We assume that the user installs Snoopy, a Trojan app that can be easily disguised as a fitness or gaming app [21]. Snoopy requires access to the motion sensors, which does not require explicit permission in Android or via CMMotionManager in iOS. Snoopy logs and periodically sends candidate extracted password events via the network. In Android this is given by the *INTERNET* permission which is classed as a normal permission, not a dangerous permission. In iOS, this is done via a normal system API and thus no additional permissions are required. The amount of data that needs to be sent is also small - a 10s batch of candidate password data is only 3 kByte, so Snoopy is unlikely to trigger any network level monitors. Note that throughout the attack, Snoopy only needs to eavesdrop motion data, without having access to any other sensing modality, such as monitoring the touch screen [32, 43].

3.2 Attack Goals

There are two key goals of the attack. The first is to successfully harvest candidate password events for later extraction. This is because sending raw motion data without segmentation will cause heavy traffic loads and lead to the app more likely being flagged. The second is to be able to infer both tapped PINs and swiped APLs. Once comprised, an attacker can unlock the physical device, accessing all stored information. Alternatively as we observe in our user study (Sec. 7), many users would use the same passwords across different accounts and platforms, where compromising one password can be harmful to many services.

3.3 System Architecture

In this section we present the high-level architecture of *Snoopy*, a system for inferring PINs and APLs on smartwatches. Snoopy contains a client front-end which runs locally on a victim's smartwatch which periodically

sends motion data to a password inference back-end that resides on the cloud. Fig.3 shows the the architecture of the proposed system.

Front-end Password Input Extraction: The front-end of Snoopy disguises itself as a harmless app, such as fitness app, and runs in the background continuously once installed. It listens to the IMU sensors and tries to detect when users are tapping or swiping passwords on their watches. To avoid being flagged as malicious by the host OS, the front-end of Snoopy uses an adaptive motion sensing strategy. It continuously samples the accelerometers at low rates to detect potential password input events. This conserves power, as accelerometers are typically one to two orders of magnitude more power efficient than gyroscopes. Once a candidate event has been detected, it enables the gyroscope and increases the sampling rate of both sensors, logging motion data until the user finishes entering passwords. Then the segment of data is smoothed and passed through a lightweight classifier, to determine retrospectively if it corresponds to a true password input event, or other user interactions such as swiping down to check notifications. In the latter case the data segment is simply discarded, while the data of true password input events is transmitted to the back-end for further analysis.

Back-end Password Inference: Given extracted segments of motion data, the back-end of Snoopy aims to infer user entered passwords. Instead of relying on bespoke signal processing algorithms which require hand-crafted features and tuning, Snoopy considers an end-to-end deep learning approach, which takes the raw motion data as the input, and computes the most likely password that the users has entered. To achieve this, Snoopy extends standard deep Recurrent Neural Networks (RNNs) to capture the unique characteristics of device motion induced by tapped PINs and swiped APLs. For PINs, it uses a hierarchical RNN with two layers to filter out the motion gaps (i.e. when the user lifts her finger off the touchscreen in between two taps) before inference, while for APLs it considers a bidirectional RNN to model the long continuous motion caused by swiping patterns. Now we are in a position to present the proposed Snoopy system in more detail.

4 IN SITU PASSWORD EXTRACTION

From a high level point of view, Snoopy infers the users' passwords by collecting and analyzing the motion data generated on their smartwatches. Of course one can task the motion sensors continuously at high sampling rates, and stream the sensor data to the cloud for password inference. However in practice, this will incur significant cost in energy, computation and communication, where the smartwatch operating systems (Android Wear or WatchOS) can easily detect such unusual behaviour and kill Snoopy instantly. To make our attack realistic, we would like Snoopy to be as "benign" as possible, i.e. it should not ask for excessive battery or bandwidth use most of time, but only become active (processing/transmitting) when the users are actually inputting passwords. In Secs. 4.1 and 4.2, we first explain how to adaptively task the motion sensors to detect potential password input events without incurring heavy load on the system. Then, in Sec. 4.3 we discuss how to extract the precise segments of motion data corresponding to those detected candidate events, and identify if the segments are related to actual password input events.

4.1 Adaptive Motion Sensing

Snoopy uses the onboard accelerometers to detect potential password input events, since they are very power efficient compared with gyroscopes [41]. Concretely, we consider an adaptive sensing strategy, which switches between three modes: *passive listening*, *password input monitoring*, and *motion data extraction*, depending on different user behaviour. Most of the time Snoopy stays in the passive listening mode, where it only samples accelerometer data at low rates and runs a gesture detection algorithm. Note that in this mode, Snoopy won't necessarily incur extra load on the sensors, since in practice major smartwatch platforms have their own gesture recognition or fitness services running in the background, which already task the accelerometer continuously. When it detects a user's intention to interact with their device, via detection of a characteristic wrist movement,

Snoopy transitions into the password input monitoring mode, where it increases the accelerometer sampling rate to look for potential password input events. It keeps analysing the received acceleration data, seeking to detect when the user will start entering their password. Once such an event is detected, Snoopy immediately turns into the motion data extraction mode, and samples both accelerometer and gyroscope at higher rates until it detects that the user has finished typing/swiping passwords. This segment of motion data is cached locally and passed through a classifier which decides if it corresponds to normal tapping/swiping, e.g. check email notifications, or a true password input event. In the latter case, the cached data is sent to the cloud for password inference. At the end of this process, Snoopy goes back to passive listening. In this way, Snoopy only actively processes and transmits short bursts of password related motion data, and avoids unnecessarily alerting the OS or malware monitoring frameworks.

4.2 Password Input Event Detection

As discussed above, when the users try to interact with their watches, Snoopy increases the accelerometer sampling rate and starts to check if any password is entered. Given the raw acceleration stream, Snoopy uses a sliding window of length T and stride S (both expressed in terms of samples) to segment the data into *frames*. Each frame contains T data points and the overlap between adjacent frames is $T - S$ (assuming $T \geq S$). In practice, the optimal T and S depend on the accelerometer sampling rate and can be learned from the data. For instance in our experiments, when the accelerometer rate is set to 40 Hz, the best T and S are 60 and 6 measurements respectively. Then for each frame, we would like to decide whether the user starts to input passwords within that frame. To achieve this, we first extract various features of the data frame, e.g. moments, maximum/minimum, skewness, kurtosis of individual acceleration axis, and different norms (e.g. l_1, l_2, Infinity and Frobenius norms) across all three axes. In the current Snoopy implementation we consider 41 features in total. Based on the extracted feature vector, we consider a Support Vector Machine (SVM) to label if the current frame belongs to a password input event. If so, Snoopy switches to the motion data extraction mode, which samples both the accelerometer and gyroscope at a high rate (e.g. 200Hz) and caches the data locally. This continues until it observes a sequence of consecutive frames that are not labelled as password input. In this way, Snoopy tends to save the motion data of as many potential password input events as possible. In what follows, we show how to extract the true password input event through sequence alignment and classification.

4.3 Frame Smoothing and Password-positive Sequence Identification

Given a cached sequence of data frames, which correspond to a potential password input event, Snoopy needs to decide: a) the accurate starting and ending frames of this event, and b) if this candidate event is a true password input event or not. For the former task, we use a smoother to align the sequence of frames based on labels of nearby frames. The intuition is that labels of adjacent frames should be consistent, i.e. a chunk of frames should either belong to a password input event or not, but not have many interleaving labels. In Snoopy we consider two types of smoothers, one based on a Hidden Markov Model (HMM) to exploit the temporal correlations, and the other based on moving average (essentially majority voting). From the output of the smoother, Snoopy extracts the longest segment of frames whose labels are positive. If the segment length exceeds a minimum threshold, Snoopy considers this segment of motion data to be able to cover the potential password input event precisely. In Sec. 6 we will show why this smoothing process is crucial, and how the two smoothers perform in different settings. However in practice, the motion data extracted might not always correspond to password input; for instance, it could correspond to users tapping or swiping their smartwatches to preview email, or check upcoming calendar notifications. Therefore, given the extracted motion data, we need to identify whether it is corresponding to a true password input event, or not. Snoopy addresses this by post-hoc feeding the extracted data segment into a binary classifier, which is trained on a pre-collected motion dataset covering various user interactions.

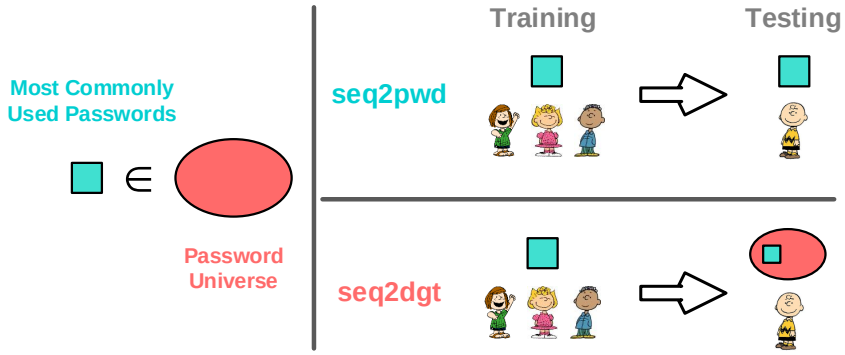

Fig. 4. Comparison of seq2pwd and seq2dgt models in Snoopy. Both models are able to attack users outside the training cohorts. In terms of password coverage, seq2pwd model can infer passwords seen before, while seq2dgt model is able to infer any password including those not encountered before.

In our experiments, we find that this classification step can be efficiently run on the smartwatches in real-time. Therefore, Snoopy is able to locally identify and extract the precise segments of motion data corresponding to password input events, and only send such data to the cloud for further password inference, which will be discussed in the next section.

5 DEEPLY LEARNED PASSWORD INFERENCE

As discussed in the previous section, the front-end of Snoopy runs locally on the users' smartwatches in the background, and eavesdrops the motion data (i.e. acceleration and gyroscope data) when users type or swipe their passwords. The extracted data segments corresponding to those passwords are transmitted to the cloud, where the back-end of the Snoopy system tries to infer the contents of the passwords. Snoopy has two inference models: sequence2password (seq2pwd) and sequence2digits (seq2dgt). Both models adopt a novel deep learning based password inference approach, which does not rely on accurate keystroke segmentation or handcrafted features, and is able to infer passwords reliably across different users and devices. In the following, Sec. 5.1 first explains how we cast the problem of password inference into a classification problem. For interested readers, the background on recurrent neural networks and their use for sequence modeling is given in the appendix. Secs. 5.2 and 5.3 describe the design of two novel deep RNN models to infer the passwords from the captured motion data.

5.1 Password Inference via Classification

We consider the task of password inference as a *classification* problem, where the category labels are a set of passwords P, i.e. four digits PINs or APLs on 3×3 grid. Then given the segment of motion data, the problem of inferring the password that the user has just input becomes that of finding a label within the database P, which can best explain the observed motion data. The size of the database P determines the inference model. Though the universe of all APLs and PINs is very large, the distribution of the adoption of them in real-world is skewed. For instance, according to statistical studies [22, 38], certain APLs tend to be more popular than others, and people only use a small set of passwords due to their bias. This means that one can utilize the skewed distribution and develop their database P targeting at the most commonly used passwords, which is more efficient and more cost-effective. The seq2pwd model in Snoopy is designed in this context. As in [5], by taking inputs as the motion data, the proposed seq2pwd model classifies a sequence to the mostly likely passwords in P, without any digit segmentation.

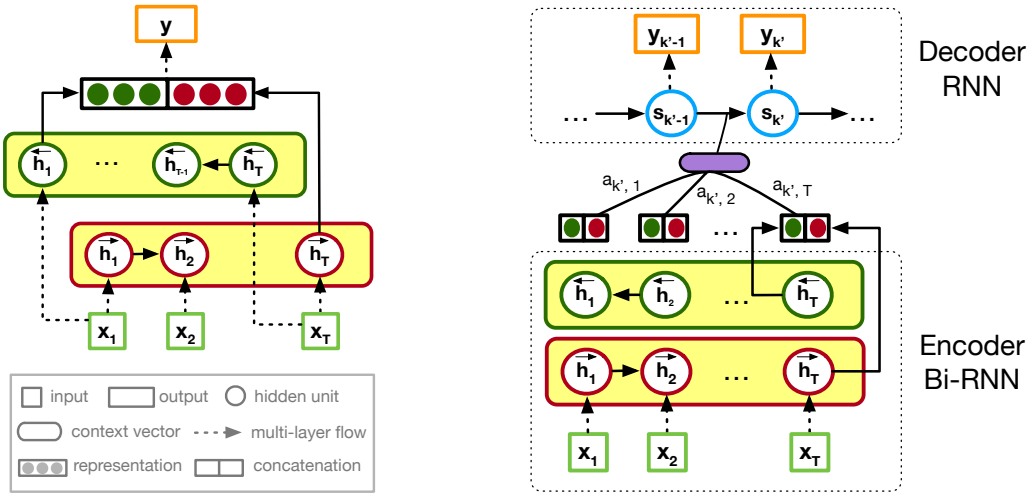

Fig. 5. The architectures of two inference models in Snoopy. Left: seq2pwd Model for commonly used password inference. Right: seq2dgt model for universal password inference.

However, despite the high likelihood of a password existing in the most commonly used password database, the expressive power of password inference is somewhat limited as seq2pwd loses its effectiveness when encountering unseen passwords ($\notin P$). And this problem gets serious when it comes to PIN inference, as the statistics of the most commonly used PINs are not as strong as the one of APLs. To solve this problem, a seq2dgt model is also proposed in Snoopy that takes inputs as motion data but predicts the password digit-by-digit. That is to say, we could train a model by a subset of the password universe but the learnt model is able to infer any member in the universe, as long as the constituted digits are seen by the model. Fortunately, there are only 10 possibilities of the digits in APLs or PINs, which is easy to meet. Therefore, the seq2dgt is essentially a digit classifier that outputs multiple predictions at each time, where the length of predictions is decided by the password length. Notably, unlike existing work, the proposed seq2dgt model does not rely on pre-processed keystroke segmentation [28] and known password lengths [5]. It automatically learns to align the chunks of motion data to the corresponding digits and learns to predict digits without knowing in advance how many there are. This is particularly useful in the case of APLs, where keystroke segmentation is not applicable [5], as the motion data of swiping APLs gives little information for digit segmentation (Fig. 1).

In the rest of this section, we introduce the seq2pwd model designed for APL inference as the distribution of popular adopted PINs is not as centered as that of APLs (see the survey in Sec.6). The seq2dgt model is proposed for both APL and PIN inference to cover the whole universe. Fig. 4 illustrates the inference coverage of the two models.

5.2 Sequence-to-Password (seq2pwd) Model for Most Commonly Used Password Inference

Swiping passwords on the touchscreen of smartwatches, the user's finger drags the device to shift around slightly, creating slip-pulse waves in the acceleration and gyroscope data (as shown in Fig. 1). This means the motion signals induced by swiping are more continuous than those of tapping, and typically without any gaps in between. Therefore, in this case the keystroke based inference approaches [29, 42] won't work well since it is impossible to segment the motion data without clear boundaries between different keystrokes.

On the other hand, for APLs the temporal correlations within the swiped pattern are much stronger, which can significantly reduce the search space and help the inference process. As discussed in Sec.2, given the current finger position, the smartwatch OS poses certain constraints on the possible directions of swipe [38], it is only possible to swipe towards three to four reachable neighbours when starting from the top left dot. Note that although the standard RNNs with LSTMs are able to capture these correlations to a certain extent, they have certain limitations. The most important is that the network only generates output from the last hidden state (i.e. $\mathbf{h_T}$). Standard problems solved by LSTMs in NLP are with the input sequences of at most 100 samples. However, the input sequence of APLs tends to be longer, and it is more difficult for the information encoded at the beginning of the input sequence to propagate through and impact the inference results.

To address this, Snoopy proposes a Bidirectional RNN (B-RNN) to model the rich temporal correlations within the input motion data. Concretely, at each timestamp k the proposed B-RNN keeps two hidden states $\overleftarrow{\mathbf{h}}_k$ and $\overrightarrow{\mathbf{h}}_k$, which incorporate the future ($k+1$, ..., T) and past (1, ..., $k-1$) information in the input sequence respectively, as shown in Fig. 5 (Left). Then B-RNN uses the same machinery as in Eqn. 7 to update those states from both directions. Unlike the standard network, it has two output nodes: one $\overrightarrow{\mathbf{h}}_T$ at the end and the other $\overleftarrow{\mathbf{h}}_1$ at the beginning. Therefore in B-RNN, information flows from both the start and the end of the input sequence, and the output of the network is generated from the concatenation of the two output variables $\overrightarrow{\mathbf{h}}_T$ and $\overleftarrow{\mathbf{h}}_1$. As shown in the next section, by using the bidirectional network architecture, Snoopy is able to preserve the long-term dependencies in the motion signals caused by swiping passwords, and thus infers passwords at much higher accuracy compared to competing approaches.

5.3 Sequence-to-Digits (seq2dgt) Model for Universal Password Inference

By formulating password inference as a sequence labeling problem, seq2pwd based RNN can guess password with a very high accuracy. However, it is difficult to adapt the seq2pwd framework to infer universal passwords. For instance, there are 389,112 possibilities for APLs and 10,000 for PINs. Using this large amount of labels for training classifiers requires huge amount of samples which is intractable in practice. We therefore propose the seq2dgt model to transform a sequence classification problem to a series of digit classification problems, where the current predicted digit conditions on the last prediction.

Despite their flexibility and power, standard RNNs can only be applied to problems whose inputs and targets share the same dimensions. It is a significant limitation in our context for two reasons. First, the lengths of APLs vary from 4 to 9, which implies the classifier needs to predict the length of digits implicitly. Fig. 6 (right) shows the duration distribution of 4 digit APLs and 7-digit APLs. As we can see, though there are 3 digits difference, the overlap of their distributions is above 40%. Second, an input IMU readings can be as long as several hundred samples, while the readings corresponding to a certain digit are only centered in a chunk of samples. Even if the number of digits is given, associating chunks to digits is difficult, as entering PINs or APLs on smartwatches *does not necessarily* occur with a uniform motion (see Fig. 1).

To solve the first problem, we resort to the encoder-decoder RNN architecture. It firstly uses an RNN as the encoder to map an input sequence to a context vector $\mathbf{c}$, and then stacks another RNN on it to decode the target sequence from the context vector. The decoder is often trained to predict the next sample $y_{k'}$, given the previous prediction $\{y_1, \ldots, y_{k'-1}\}$. Formally, the probability of output sequence Y with the length of T' (a few number of digits in our context) is defined as:

$$p(\mathbf{Y}) = \sum_{k'=1}^{T'} p(y_{k'} | \{y_1, y_2, \cdots, y_{k'-1}\}, \mathbf{X}) \tag{1}$$

With the decoder RNN, each condition probability is modeled as:

$$p(y_{k'}|\{y_1, y_2, \cdots, y_{k'-1}\}, \mathbf{X}) = g(y_{k'-1}, s_{k'}, \mathbf{c}) \tag{2}$$

where k denotes a timestep in inputs ($1 < k < T$) and k' is a timestep in outputs ($1 < k' < T'$). g is a nonlinear, potentially multi-layered function that outputs the probability of $y_{k'}$; $s_{k'}$ is the hidden state of the decoder RNN and $\mathbf{c}$ is the encoded context vector. Fig. 10 (Right) illustrate the above RNN architecture used in our seq2dgt model.

By introducing a dummy digit symbol $< EOS >$, standing for end of output sequences, the unknown lengths of APLs can be implicitly determined. In this way, we have 10 candidate 'digits' for each digit in APLs that our model needs to predict, i.e., $\{1, 2, \ldots, 9, < EOS >\}$. The digit length of an APL is decided when the seq2dgt models gives the first $< EOS >$ symbol. For instances, a prediction $'1, 2, 3, 6, < EOS >, 9, < EOS >'$ indicates the length of target APL is 4. All predicted digits after the first $< EOS >$ symbol e.g., $'9', < EOS >$ in the example, are not counted. $< EOS >$ usage is widely adopted in the field of NLP [34]. An analogous instance of ours is machine translation, where a source English sentence may not have the same number of words as its Chinese translation. The dummy $< EOS >$ symbol can prevent the model generating an infinite number of words. Note that, this step is only for APLs; a PIN's length is fixed to 4 in most scenarios.

Originally, the context vector $\mathbf{c}$ is computed by encoding all inputs. However, the second problem remains as the IMU readings are sampled at 200Hz, whose lengths are dramatically longer than a few digits but only a part of them contribute at one decoding timestep. Here we introduce the attention mechanism in our seq2dgt model. Formally, the conditional probability in this attention seq2dgt is defined as:

$$p(y_{k'}|\{y_1, y_2, \cdots, y_{k'-1}\}, \mathbf{X}) = g(y_{k'-1}, s_{k'}, \mathbf{c}_{k'}) \tag{3}$$

Unlike the conditional probability in Eq. (2), here the probability is conditioned on a distinct context vector $\mathbf{c}_{k'}$ for each output digit $y_{k'}$. The new context vector depends on a sequence of hidden sates $(h_1, \ldots, h_T)$ to which an encoder maps the input sequence $\mathbf{X}$, where we adopt a bidirectional RNN, i.e., $h_k = [\overleftarrow{\mathbf{h}}_k, \overrightarrow{\mathbf{h}}_k]$. Formally,

$$\mathbf{c}_{k'} = \sum_{k=1}^{T} a_{k'k} h_k \tag{4}$$

where $a_{k'k}$ are the weights determing the contribution of h_k in encoding $\mathbf{c}_{k'}$ for the k'-th digit and it can be determined through backpropagation in an end-to-end optimization. The attention mechanism is widely adopted in the scenario where input sequences are very long, and a single context vector is too compressed to decode outputs. For example, Hermann et al. [13] have achieved impressive results in document summarization by introducing the attention mechanism in their models, which solves the problem that the number of words of in the input documents are much larger than the ones in the output summaries. As shown in the next section, the seq2dgt model benefits from this attention mechanism and it is able to adaptively focus on specific chunks of input (with high attention weights) when generating digits in different positions of the passwords.

6 EVALUATION

We evaluate the proposed Snoopy system extensively on large-scale real world datasets collected in three different sites: *Oxford*, *Shanghai* and *Harbin*. In total, our experiments collected preferred/generated passwords from **420** anonymous participants, recruited a separate group of **362** volunteers to contribute their motion data when entering passwords on smartwatches (worn on their left hands), and accumulated over **60***k* samples of motion data during password entries[1]. In the following, Sec. 6.1 focuses on evaluating the performance of the front-end

[1]The study has received ethical approval *R50768* from the University of Oxford.

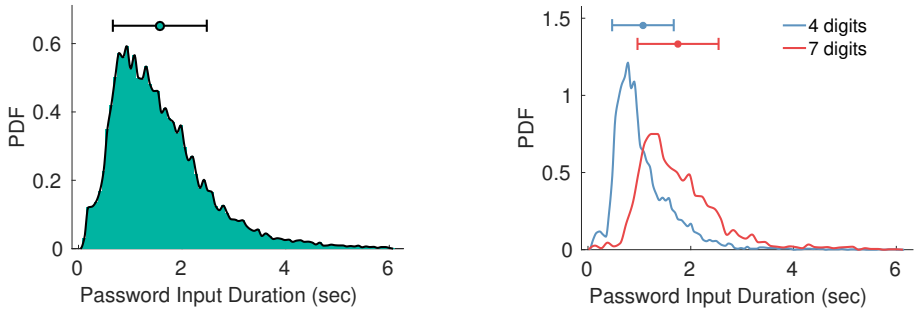

Fig. 6. PDF of APL input duration. Left: duration distribution of swiping APLs. Right: Differentiating a 4-digit APL and 7-digit APL is difficult based on their duration distribution.

password extraction capability of the proposed Snoopy system; Sec. 6.2 presents the performance of Snoopy in inferring APLs; and finally Sec. 6.3 assesses our system's capability of PIN inference.

6.1 Performance of Password Input Extraction

6.1.1 Experimental Setup.

Data Collection: To evaluate the performance of password input extraction, we recruited 15 volunteers (10 males and 5 females), and asked them to wear different smartwatches (Android and iWatches) on their left wrists. Throughout our experiments we use four different models of Android watches: Sony SmartWatch3, Samsung Gear Live, Moto 360 Sports, LG Urbane, and two Apple watches: 38mm and 42mm versions of iWatch2. Note that Android smart watches typically use APLs for system level authentication, whereas Apple watches use PINs.

During the experiments, we asked the participants to perform three different types of actions: *password input* where they enter their passwords on their smartwatches; *non-password input* where they tap/swipe on the watches screen to do other tasks (e.g. preview email or check a calendar notification) but not to enter a password; and *no input* when they just perform a series of activities wearing their smartwatches, such as drinking, drawing, eating, walking, going down/upstairs, typing on keyboards and holding hands still. We designed a data collection app on the smartwatches, which samples the motion sensors in the background (100Hz in this case), and instructs the participants to perform certain actions at a given time. In this way we can obtain accurate ground truth as to when the user is performing a certain action.

To collect rich enough data, in one episode we requested a participant to at least perform three actions, where the action in the middle should be password input or non-password input action, e.g. she may first walk, then enter her password, and finally go upstairs. Each participant is requested to contribute multiple episodes, and in total we obtained 455 episodes for Android watches, and 387 for Apple watches.

Competing Approaches: Since the front-end of Snoopy has to run locally, in this series of experiments we only consider lightweight approaches that can run in real-time on the smartwatches. Recall that the task of password extraction has not been attempted on smartwatches before. Previous related work has focused on the extraction of PINs on smartphones only [8, 28, 42]. However, features used to extract keystrokes on smartphones are not suitable for smartwatches due to the limited screen size and low signal to noise ratio (see Fig. 2). And even in the case of smartphones, previous work has assumed that any keystroke is part of a password; however in practice, keystrokes could be used in the middle of other tasks not related to password input, for example replying to email. There is no competing approach that currently addresses the entire password extraction task. In what follows, we evaluate a realistic password extraction approach for smartwatches that has three stages. In the first stage, we assess how well we can detect the beginning of a password (see Sec. 4.2 for details); we compare a number of

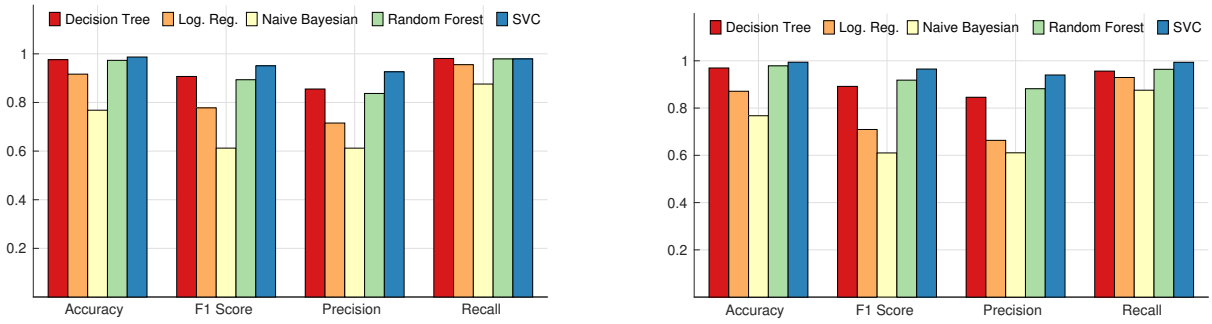

Fig. 7. Performance of detecting potential password input events. Left: PINs; Right: APLs

classifiers including the Support Vector Classifier (**SVC**), **decision trees**, **logistic regression**, **naive Bayesian** and **random forest** classifiers. Once we have detected the beginning of a password, in the second stage, we continue to define the full extent of the potential password, by classifying individual frames, and smoothing classification results (see Sec. 4.3 for details). Here for smoothing, we compare the Hidden Markov Model (**HMM**) based and the voting based moving average (**moving average**) approaches implemented in Snoopy with the baseline approach without smoothing (**raw**). Once the start and end of a potential password are found, the final stage classifies this sequence as an actual password, or a non-password sequence (as discussed at the end of Sec. 4.3); here we also compare the performance of several classifiers, including **SVC**, **decision trees**, **logistic regression**, **naive Bayesian** and **random forest**. The collected dataset is split into a training set (data from 10 subjects) and a test set (data from the other 5 subjects), and we consider 5-fold cross-validation.

6.1.2 *Experiment Results.*

Detecting Password Input Events: As discussed in Sec. 4.2, for a given frame of acceleration data, we would like to decide whether it corresponds to password input or not. As in many other binary classification problems, here we consider the precision, recall, F_1 score and accuracy of the classifiers. Fig. 7 shows that the SVC outperforms competing classifiers in terms of all evaluation metrics. For both PINs and APLs, it can achieve > 0.95 F_1 scores and > 0.98 accuracy. The random forest and decision tree classifiers can achieve comparable recall with SVC, but their precision is nearly 8% lower than that of SVC. Based on these results, in what follows, we adopt SVC as the default classifier for detecting the beginning of a potential password input event in Snoopy.

Smoothing Detected Sequences: The above SVC approach is iteratively used in ensuing frames to classify them as password-related ('1') or not ('0'). When we start seeing a lot of '0'-s this indicates the end of a potential password. The candidate password sequence that we derive (e.g. '11001111110001101') is then passed through a smoothing process as discussed in Sec.4.3. The smoother adjusts the frame labels taking into account those of nearby frames, and further refines the start and end point of the potential password input event. To evaluate the performance of the smoothing process, we consider the similarity of a frame sequence s_r with respect to the ground truth s_t:

$$d(s_r, s_t) = \frac{|s_r \cap s_t|}{(|s_r| + |s_t|)/2} \tag{5}$$

where $|\cdot|$ is the cardinality of the positive labels, and $|s_r \cap s_t|$ is the number of frames that have the same labels in both s_r and s_t. Intuitively a sequence s_r with higher d is better because it is closer to the ground truth, and thus tends to contain larger portion of correctly labelled frames. Fig. 8 shows the average similarity scores of sequences generated by different smoothers (HMM and majority voting moving average vs. no smoothing). As we can see for different frame sizes, sequences without smoothing (raw) consistently have lower d scores. This

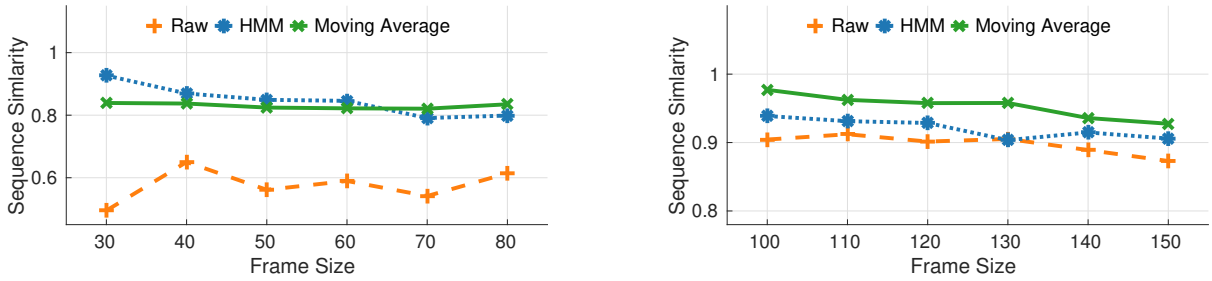

Fig. 8. Performance of sequence smoothing when using different frame sizes. Left: PINs; Right: APLs

	F1 Score	Precision	Recall
Decision Tree	0.89	0.92	0.86
Naive Bayesian	0.91	0.90	0.91
Logistic Regression	0.95	0.96	0.93
SVC	0.93	0.92	0.95

Table 1. **PIN** sequence identification results.

	F1 Score	Precision	Recall
Decision Tree	0.93	0.91	0.93
Naive Bayesian	0.91	0.90	0.91
Logistic Regression	0.96	0.97	0.95
SVC	0.94	0.95	0.94

Table 2. **APL** sequence identification results.

ave./max (%)	Feature Extraction	SVM
Sony SW3	8.9/23.0	7.2/21.4
Samsung Gear Live	10.9/25.2	9.8/18.5
Moto 360 Sports	10.5/22.7	10.1/25.5

Table 3. CPU load of running feature extraction and SVM.

confirms that the smoothing process is beneficial. For instance, for APLs, the moving average smoother can reach 0.98 in terms of sequence similarity score, while for PINs the HMM smoother can achieve 0.94. Note that for the PINs, the performance gain between not using (raw) and using smoothers (HMM and moving average) can be up to 35%. This is because the motion data generated by PINs contains "gaps" between two adjacent finger touches, where the detection approaches would naturally label frames within the gaps as negative (i.e. no password input). In those cases, the smoothers can correct those errors, and output a sequence with more consistent labels. It is also interesting to see that although the two smoothers considered in Snoopy have comparable performance, HMM tends to work better than moving average for PINs, but can be inferior for APLs. Again this is because HMM is able to mitigate those non-informative gaps within data of the PINs, while for APLs moving average is more robust.

Password-positive vs. Password-negative Sequences: Given a smoothed sequence, the front-end of Snoopy needs to identify if it corresponds to an actual password input, or non-password input such as swiping to check notifications. As discussed in Sec.4.3, Snoopy addresses this by feeding an entire sequence into a binary classifier. Tables 1 and 2 show the identification performance of different classifiers for PINs and APLs respectively. We see that unlike the more expensive random forest and SVC, the simpler classifiers work surprisingly well for this task. Note that for both types of passwords, the simple classifiers can achieve up to >0.95 F_1 score, which means they are able to reliably distinguish motion caused by password input from that of other interactions. This is because due to the small size of the smartwatches, the ways to interact with their touchscreens are very limited. Therefore the motion signals of PINs or APLs are very unique compared to others.

Resource Consumption: The final set of experiments analyse the resource consumption of Snoopy front-end on three Android watches with different hardware specifications. Tab. 4 shows the average/maximum CPU load, delta current consumption and battery usages when the front-end is running the following three tasks: a) password

input detection (Sec. 4.2); b) password-input data smoothing and identification (Sec. 4.3); and c) uploading extracted data to the back-end. We see that among the three tasks, uploading actually consumes the largest amount of energy, while detection and smoothing are relatively cheap. On the other hand, detection and smoothing tend to occupy the CPU more than uploading. This is expected because it is well known that transmitting over WiFi is power-consuming on smartwatches, while detection and smoothing are more computation-intensive as they involve running SVM classifiers. More specifically, as shown in Tab. 3, on all three watch platforms, running feature extraction and SVM classifiers consume similar level of CPU resources (~10%), but the former is slightly more expensive since it involves continuous caching operations, e.g. maintaining the sliding windows. In addition, like many other apps [20], to minimize impact on battery lifetime, Snoopy only uploads cached data when the watches are connected to power with WiFi connections available. As shown in Tab. 4, in general the Snoopy front-end doesn't require excessive resources, and when disguised as an innocent fitness app, it is not likely to have noticeably abnormal energy/computation impact on the smartwatches.

Model	SoC	RAM	Battery Cap.	Task	CPU load (avg/max)	Current delta	Battery Usage
Sony SW3	Qualcomm APQ8026 SD 400	512MB	420 mAh	Detection	9.2%/17.5%	8.9 mA	2% per hr
				Smoothing	7.2%/15.2%	3.1 mA	
				Uploading	2.4%/9.9%	18.9 mA	
Samsung Gear Live	Qualcomm MSM8226 SD 400	512MB	300 mAh	Detection	8.1%/19.4%	11.4 mA	3% per hr
				Smoothing	15.6%/29.3%	6.2 mA	
				Uploading	2.1%/19.3%	25.1 mA	
Moto 360 Sports	Qualcomm MSM8926 SD 400	512MB	300 mAh	Detection	8.3%/26.2%	16.1 mA	3% per hr
				Smoothing	7.3%/22.4%	3.8 mA	
				Uploading	2.3%/26.1%	22.3 mA	

Table 4. Resource consumption (CPU and power) of the Snoopy front-end on smartwatches with different hardware specs.

6.2 Performance of APL Inference

We are now in a position to turn our attention to how the back-end password inference component of Snoopy performs. In this section we firstly discuss the performance of APL inference, while the PIN inference will be covered in Sec. 6.3.

6.2.1 Experiment Setup.

APL Database Construction: As discussed in Sec. 5, to infer the user entered APLs, both the seq2pwd and seq2dgt models considered in Snoopy require a good password database P for training, which can cover as many common passwords as possible. To construct such a database P, we consider the publicly available APL data reported in [22] and also collected our own dataset. The APL dataset in [22] contains ~4,000 APLs entries collected from the anonymous users (with duplications). From this dataset, we rank the distinct APLs according to their frequencies, and select the most popular 113 APLs that can cover half of all the APL entries (2000 out of 4000). This ensures that the selected APLs can achieve a good coverage of the most commonly used APLs, while leaving out those APLs that are seldom used.

We also recruited 112 anonymous participants to survey their preferred passwords (both PIN and APL) when using mobile devices. The purpose of collecting our own password dataset is to obtain an independent dataset in addition to the publicly available data, which would make the constructed password database P more diverse. During the data collection process, we have made sure that every step complied with data privacy policies, and there is no link between the collected data and any individual participant. In particular, we have first obtained

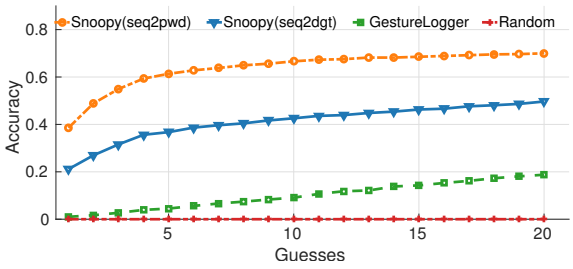

Fig. 9. APL inference accuracy of competing approach and two proposed models in Snoopy.

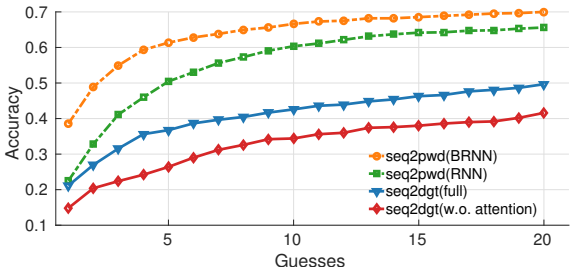

Fig. 10. Impact of network architectures on the inference accuracy.

the participants' consent that their data will be used in a scientific study to evaluate password security on smartwatches. If a participant agreed to proceed, she was then given an Android watch, and we asked her to wear the watch on her left wrist. Then the participant is provided with an instruction sheet, which asks her to set a password in a survey app on the smartwatch. The survey app only records the entered passwords by monitoring touches on the touchscreen. When the participant finished entering the password, it asks if she is aware of the purpose of the study and would like to contribute this password. If so, the password is assigned with a unique random ID, and written into a random line of a local text file on the smartwatch. Otherwise there is no information saved. Note that during this process, the participants were asked to input passwords in private and take their time. The watches and instruction sheet were passed directly to the next participant without our intervention. After the survey process, fortunately we obtained 112 APL entries from all participants, among which we have extracted 64 distinct APLs. Finally, we fuse those 64 surveyed APLs and the 113 APLs extract from the existing dataset [22], and construct a password database P with 147 *distinct* APLs.

APL Input Motion Data Collection: Given the above constructed APL database P, we recruited a total number of 322 participants across three experiment sites to collect the motion data when they are entering APLs on their smartwatches. Each participant was randomly given 6 APLs selected from P. The participants were asked to wear the smartwaches on their left wrist but in the most comfortable way, and then enter each password in our data collection app about 20 times. The app logs the ground truth by monitoring tap/swipe on the smartwatch screen, and saves the motion data at the same time. In total, we have collected 36, 569 valid samples, each of which contains an APL and the motion data when it was entered. This set of data is used to train our models in Snoopy.

Competing Approaches: We implement both the seq2pwd and seq2dgt models considered in Snoopy using Keras [9], and train them on NVIDIA K80 GPUs with the Adam optimiser [16]. To the best of our knowledge, Snoopy is the first work to study the problem of inferring smartwatch APLs, and there is no existing work that can infer APLs without knowing the exact segmentation of digits within APLs (as discussed in 5.3). Therefore, here we only consider one of the best APL inference approach designed for smartphones, **GestureLogger** [5], which bears some resemblance to the proposed seq2pwd model in Snoopy.

6.2.2 *Experiment Results.*

Field Test APLs vs. Constructed APL Database: The first experiment verifies the representativeness of the constructed APL database. We recruited an independent cohort of 308 volunteers (115 female and 193 male, mean age 39.8 with σ = 11.3, Mdn = 40, ranging from 18 to 63), and made sure that none of them was involved in building the APL database. Then we asked them to conduct an anonymous online survey to provide their preferred APLs. This survey also complies with data privacy policies and there is no link between the collected APLs and any individual participants. After the survey process, we obtained 308 APL entries from all participants. We found that among the 308 APL entries, 223 (72.4%) fall into the constructed APL database. This confirms that

the constructed P indeed covers a good variety of commonly used APLs, and it is possible to use P to accurately infer the user entered APLs.

APL Inference Accuracy in Field Test: This experiment evaluates the performance of APL inference of the proposed Snoopy system in the field test. As discussed above, Snoopy uses the constructed password database P and the associated motion data to train its models. To evaluate its true capability of inferring APLs in real-world scenarios, we consider the field test APLs which are independent with the APL database P. Concretely, we consider a similar approach as in [43], and recruited another 20 volunteers (13 males and 7 females), who hadn't contributed any password or motion data, to reproduce (i.e. input) the 308 APLs obtained from the filed test. The motion data associated with APL entries was collected using the same watch app, and on average each volunteer swiped about 120 APLs. Eventually we obtained 2,368 valid samples using three different types of watches (Sony SW3, Samsung Gear Live and Moto 360), and this data is the used to assess the accuracy of APL inference.

We consider the successful rate at different number of attempts [25, 38] as the metric inference accuracy, which has been widely used to quantify the threat level of a malicious app [12]. As in GestureLogger [5], we set the maximum possible number of attempts to 20.Both proposed and competing methods take as input a motion signal sequence, and return scores for different candidate passwords. We then select the top 20 passwords, which are the most likely passwords according to the technique used. The first guess always selects the top password, the second guess the next most likely, and so on.

Fig. 9 shows inference accuracy of APLs, where we include random guess as the naive baseline. We see that both of the proposed models (seq2pwd and seq2dgt) in Snoopy consistently outperform GestureLogger, achieving up to 3-4 fold improvement in inference accuracy. In particular, if only allowed to guess once, seq2dgt model can get 21% accuracy, i.e. one in five times it is able to guess the correct APL, while seq2pwd can achieve an even higher accuracy of 39%. We found that although seq2pwd model can only predict APLs within the constructed database P ($|P| = 147$), its inference accuracy is 'worryingly' good: if 10 guesses are allowed, its accuracy can be 65% and increases up to to 68% for 20 guesses. Note that here the inference is performed on the field test data which is completely independent from the data used to construct P. This means that the APL database P constructed in our experiments is very representative, and thus in practice, it is possible to infer most of the popular APLs with such a database P. In addition, although GestureLogger also infers APLs from P, its accuracy is very limited and only able to reach 19% after 20 attempts (more than 3 folds lower than seq2pwd).

On the other hand, seq2dgt is not limited to the size of database P, and can predict any APLs within all the 389,112 possibilities. We see that although the search space now is $\sim$ 2700 times bigger, seq2dgt can still achieve decent inference accuracy: about 43% after 10 attempts and up to 50% with 20 guesses. This indicates that the proposed seq2dgt model can indeed learn the underlying mechanism of user entering APLs, and make informed predictions when applicable. Note that although seq2dgt solves a much more challenging problem, i.e. no prior knowledge on popular APLs or perfect segmentation between digits, its accuracy is still way superior than the state-of-the-art GestureLogger: within 20 guesses, seq2dgt is 250% more likely to hit the correct password than GestureLogger.

Impact of Network Architecture: This experiment investigates the inference performance of Snoopy when using different deep network architectures. For seq2pwd model, we compare the inference accuracy of the proposed bi-directional RNN (B-RNN) and standard RNN. As shown in Fig. 10, B-RNN is about 15% superior to standard RNN at the first attempt, and is $\sim$ 8% more accurate on average within 20 attempts. This means that the temporal correlations within APLs are difficult to be captured by standard RNNs, and by allowing gradient flow from both directions, B-RNN is able to capture richer information especially in long APL sequences. On the other hand, for the seq2dgt model, we see that the proposed attention mechanism can improve about 10% of inference accuracy over the standard architecture. This is because the attention mechanism helps the network to focus more on the chunks of informative sensor readings, i.e. when finger tips slide through digits, while the standard network only decodes APLs based on fixed context vectors.

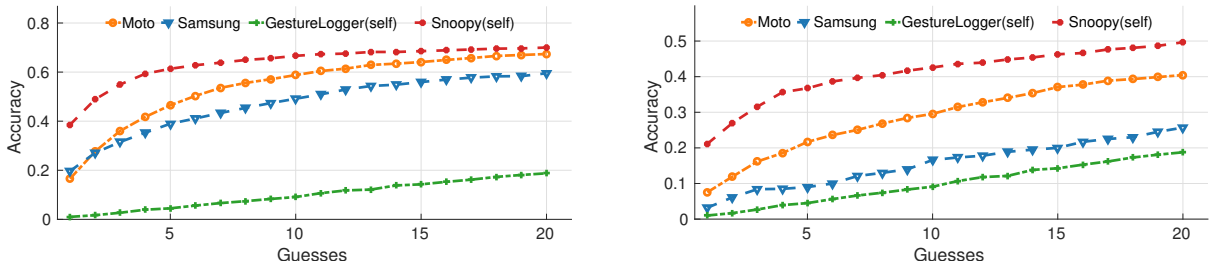

Fig. 11. Cross-device APL inference accuracy. Left: seq2pwd model; Right: seq2dgt model.

Inference Accuracy vs. Device Heterogeneity: In previous experiments, although we always consider cross-user inference, i.e. our system is trained on data collected from one group of users, but tested against data generated from the others, we assume that the same model of device are used in training and testing. For instance, to infer passwords entered on a Sony SW3 watch, we assume that Snoopy can be trained with data collected on Sony SW3 watches (not necessarily the same one). In this experiment, we further push the limit, and see how Snoopy performs in the presence of device heterogeneity. This is very challenging, since the watches used for testing may have different sensors, dimensions or shapes (round vs. square) with those used for training. To demonstrate this, for APLs we train Snoopy with data from Sony SW3 watches, and test it on the other two models, Samsung Gear Live and Moto 360 Sports respectively. Fig. 11 shows the inference accuracy with the same device model, and across difference models. Note that here we put the performance of GestureLoger (trained and tested on the *same* device model) as the baseline. As we can see, for seq2pwd, when tested on different devices, its performance drops elegantly. For Moto 360 which has round shape (the Sony watches used for training are square shaped), the performance only decreases by ~13% on average. This means heterogeneity in the shape of smartwatches won't affect password inference performance significantly. On the other hand, the performance on Samsung Gear Live (square shaped) drops about 20%. Note that even for this worst case, the inference accuracy of seq2pwd can still reach ~50% after 10 attempts, while the best competing approach GestureLoger is less than 10%. On the other hand, from the right of Fig. 11, we see that seq2dgt model is slightly more sensitive to device heterogeneity. On Moto watches the accuracy decreases ~ 18% while about 25% on the Samsung watches. This is also expected since seq2dgt works against a massive search space (389, 112 possibilities), where a small perturbation in sensor readings might lead to very different predictions. However even in this challenging case, the inference accuracy is still consistently higher than that of GeastureLogger, which is trained and tested on the same device models.

6.3 Performance of PIN Inference

In this section, we further evaluate the performance of the proposed Snoopy system in inferring PINs entered on smartwatches.

6.3.1 Experiment Setup.

Training Data Collection: Like the previous APL case, we first constructed a PIN database by surveying the same 112 anonymous participants. We follow the same data collection protocol as discussed in the previous section, and obtained a database P containing 79 distinct PINs from the 112 responses. To collect the PIN input motion data, we recruited a group of 156 users, and ask them to input 6 randomly selected PINs from P with smartwatches (iWatches) worn on their left wrists. We use a similar data collection app, and eventually collected 23, 144 valid samples of motion data associated with the PIN input events. As in the APL case, this data is used to train the proposed Snoopy system.

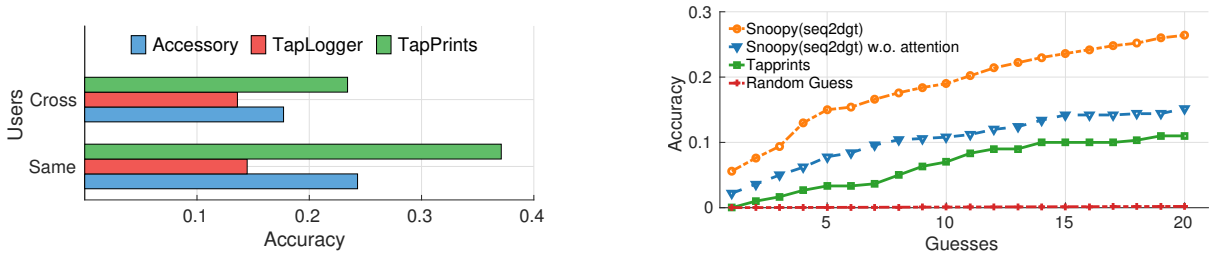

Fig. 12. Performance of PIN inference. Left: element-wise accuracy of existing approaches; Right: inference accuracy of Snoopy and competing approaches.

Competing Approaches: To the best of our knowledge, there is no existing work studying PIN inference on smartwatches. Therefore, we compare Snoopy with the state-of-the-art PIN inference approaches designed for smartphones. These approaches usually adopt an element-wise inference: it firstly identifies each digit of the PINs and then concatenate the identified elements into whole passwords. We implemented three well-known approaches: 1) **Accessory** [29]: which uses a random forest classifier to identify the individual tapped digits from accelerator data; 2) **TapLogger** [42] which is very similar to Accessory but uses k–NN classifier; and 3) **TapPrints** [28], which considers both acceleration and gyroscope data, and uses an ensemble classifier (SVC, decision tree, logistic regression and random forests as base learners) to detect PIN elements. Details of the competing approaches can be found in Tab. 6. Note that all of the three competing approaches require prior knowledge on the accurate segmentation of motion data, while Snoopy is able to perform end-to-end inference.

6.3.2 Experiment Results.

Field Test PINs vs. Constructed PIN Database: The first experiment is to evaluate to what extend can the constructed PIN database cover the commonly used PINs in the real world. We obtain from [3] a large online surveyed PIN dataset containing 204, 508 PINs, and consider it as the field test PIN data. As in the APL case, we rank those PINs according to their frequencies, and then select the top 642 distinct most popular PINs to cover half of all the PIN entries ($> 100k$). For those 642 distinct PINs, we compare them with those in our PIN database P. We found that unlike the APL case where we observe a significant overlapping between the field test passwords and the constructed password database, here the overlapping is only about 7%. Unfortunately to the best of our knowledge there is no study so far that can provide thorough explanation on this. Our intuition is that people often use meaningful numbers to themselves as PINs, such as birthdays or addresses, which are quite unlikely to collide. In addition, clearly there is less constraints when tapping digits on touch screen than that of swiping APLs, and thus people may tend to choose from those easy-to-swipe APLs. Based on this observation, in the following we only consider *seq2dgt* for PIN inference but not seq2pwd, since the latter can only predict PINs from the database P which only covers a small percentage of commonly used PINs.

Element-wise Inference Accuracy: Before evaluating Snoopy, in this experiment, we first evaluate the performance of the existing element-wise inference approaches. We consider two types of password inference. Firstly, the *same-user inference* assumes that the algorithms would infer passwords entered by a user *with* access to the ground truth password-input motion data of this particular user, e.g. they have previously "seen" the user entering passwords (i.e. knowing the password contents), and collected the corresponding motion data. On the other hand, the *same-user inference* assumes the algorithms have to infer a user's passwords *without* access to her previous labelled motion data. As shown in Fig. 12 (Left), the performance of element-wise approaches is very limited: the best algorithm can only achieve about 25% accuracy when inferring a single digit for a PIN (one time guess), while the accuracy of random guess is 1/10. Even in the most favorable case where the testing objects

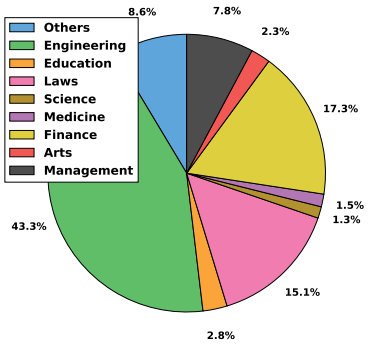

Fig. 13. Background distribution.

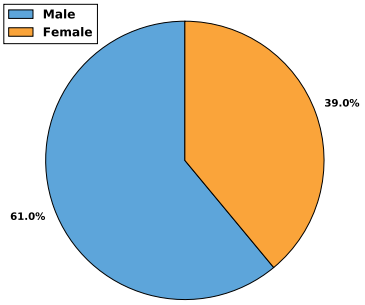

Fig. 14. Gender distribution.

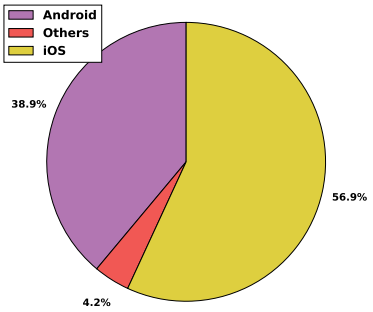

Fig. 15. Platform distribution.

are the same users in training, the performance only grows ~ 10%. A possible reason for the low segmentation accuracy is that the SNR of the motion data on smartwatches is much lower than that of smartphones, which limits the performance of prior art designed for smartphones significantly. Overall TapPrints outperforms the other two, and thus in the following experiments, we only include TapPrints in our competing approaches.

PIN Inference Accuracy in Field Test: To evaluate the inference accuracy of the proposed Snoopy and competing approaches, we firstly collected a PIN input motion dataset based on the 642 distinct PINs obtained from the field test. As in the previous section, we recruited an independent group of 20 participants who hadn't contributed any data to enter those PINs on their smartwatches. The mean age of participants is 32.3 (σ = 10.4, Mdn = 31, ranging from 18 to 53), and the data collection process is similar to that in the previous APL case. We compare the performance of the proposed seq2dgt approach in Snoopy (with and without attention mechanism) and the best competing algorithm TapPrints on this dataset, and include the naive random guess as the baseline. As shown in Fig. 12 (Right), both variants of Snoopy(seq2dgt) consistently outperform the TapPrints, and is able to achieve > 2× accuracy improvement. In particular, with a single chance Snoopy is able to achieve 6% success rate, which is much higher than random random guess (0.01%). If more attempts are allowed, Snoopy can achieve up to 18% success rate after 10 guesses and 28% within 20 attempts. The performance of the competing TapPrints is much lower, and can only make to 11% after 20 attempts. In addition, we see that in this case the attention mechanism provides more performance gain (up to about 10%) comparing to that in the previous APL case. This is because in the case PIN inference, the motion data associated with gaps between two taps is mostly noise, which won't provide any useful information for prediction. Therefore by using the attention mechanism, Snoopy can effectively ignore those gaps by assigning dynamic weights during decoding, i.e. it would put more weights on the data segments associated with real taps.

7 USER STUDY

We complement our experimental evaluation of Snoopy with a user study, which aims to understand the users' awareness of the potential password leak on smartwatches via motion data and its consequences. We distributed questionnaires on social media in the UK and China, asking anonymous participants to provide basic demographic information such as gender, age, occupation and smartwatch platforms used. The questionnaire consists of the following yes-or-no questions:

- Q1: Have you used the same passwords in different accounts/platforms, e.g., same PIN for PayPal[2] and your device screen lock?

[2] https://www.paypal.com/gb/home

- Q2: Did you know (before this survey) your passcodes on smart devices could be leaked through motion sensors?
- Q3: Have you allowed or will allow third-party apps on your smart wearable devices to access your motion data, e.g., allowing WeChat+[3] to record the number of steps you've walked?
- Q4: Have you disabled or considered disabling third-party apps monitoring your motion data when entering passwords on your smart device?
- Q5: Do you often type on smartwatches (> 3 times a day), e.g., sending instant messages[4] or editing emails[5]?
- Q6: Are you a smartwatch owner?
- Q7: Have you set up unlock passwords on your smartwatches?

From two periods, we received 745 anonymous responses for the first 5 questions and 301 anonymous responses for the last two questions respectively. Among them we have users from nine different occupation categories (Fig. 13), with slightly more male than female (61% vs. 39% as in Fig. 14), and an average age of 32.3 (σ=12.1, Mdn=28, ranging from 18 to 63). This is expected since in general, young male users are more willing to try out new gadgets such as smartwatches. We also observe that Android and iOS dominate the smartwatch market. As shown in Fig. 15, 39% and 57% of our participants wear Android or Apple watches, while only a tiny percentage (4%) of them were using other platforms.

Fig 16 summarizes the distribution of answers. First of all, we see that about 25% of our participants are smartwatch owners, and this number is expected to grow rapidly in the near future according to [30]. Among those smartwatch owners, we find that the majority of them (73%) would use unlock passwords for their devices. This indicates that smartwatches are truly becoming pervasive, and users tend to rely on the built-in unlock passwords (APLs or PINs) to protect their devices. Another key finding is that over 80% of the participants tend to use the same password across different applications. This means that the consequences of leaking smartwatch passwords can be significant: what if the smartwatch PIN is the same PIN used for online banking or Paypal? We also observe that a large number of users (>60%) did not know that the motion sensors on smart devices may leak sensitive information. In fact, 76% of the participants would allow, or had already allowed, third-party apps to access their motion data, and less than 20% of them would consider disabling motion sensors when entering sensitive information on their devices. This shows although motion sensor attacks have been extensively studied

[3] https://www.wechat.com/en/
[4] https://hangouts.google.com/
[5] https://sparkmailapp.com/

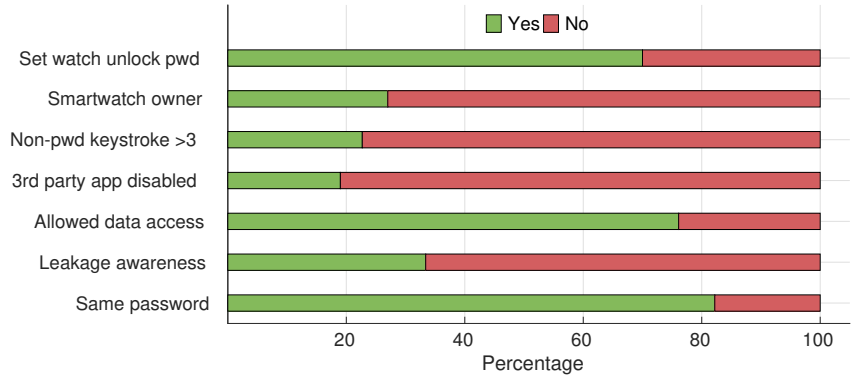

Fig. 16. Survey results. They were asked 8 questions about smartwacth usage and password settings.

Sampling Method	Energy Consumption	APL		PIN
		Accuracy (seq2pwd)	Accuracy (seq2dgt)	Accuracy (seq2dgt)
50 Hz (const)	2.0% per hr	56.1%	25.7%	9.1%
100 Hz (const)	4.4% per hr	61.5%	34.6%	14.4%
200 Hz (const)	6.3% per hr	66.3%	44.5%	23.4%
Snoopy (adaptive)	2.1% per hr	64.5%	42.6%	18.0%

Table 5. Energy consumption and inference accuracy with different motion sensing strategies (profiled on Sony SW3 watches). For the first three approaches (50Hz, 100Hz and 200Hz), we constantly sample the motion sensors and directly feed the data to the inference algorithms. The proposed Snoopy uses an adaptive sensing approach, which uses feature extraction techniques to only sample high frequency data when the users are entering passwords.

in academia, most users are still not aware of this. In addition, we see that unlike smartphones, users seldom perform complex interactions on smartwatch screens such as typing, since the size of them is much smaller than phones. This means in practice it is easier to detect password input events from other tapping/swiping on smartwatches, which makes this a more vulnerable attack surface.

8 DISCUSSION

This section discusses some important issues related to the proposed Snoopy system. In Sec. 8 we show how different motion sensing strategies have knock-on effects on both energy consumption and password inference accuracy, and further highlight the benefit of the Snoopy front-end. In Sec. 8.2 we discuss the limitation of the proposed approach, and provide two potential mitigations for such attacks.

8.1 Sampling More vs. Sampling Smart

Energy Consumption: As discussed above, the front-end of Snoopy considers an adaptive motion sensing approach, which uses feature extraction techniques and SVM classifiers to detect when the user are entering passwords on their devices. Of course running feature extraction and SVM on top of motion sensing requires extra energy, however, as shown in the first two columns of Tab. 5, the overhead is not excessive: the energy consumption of Snoopy front-end is very similar to that of constantly sampling 50Hz motion data, and is only 1/3 of the 200Hz approach. Note that the adaptive sensing mechanism in Snoopy constantly tasks accelerometers at 40Hz, and triggers both accelerometer and gyroscope at 200Hz when the classifier fires. This confirms that comparing to sampling motion data at high freq, the extra energy consumption required by Snoopy front-end is cheaper.

Password Inference Accuracy: We would also like to investigate the impact on password inference accuracy of different motion sensing strategies. Here we consider the inference results within 10 attempts. Obviously sampling higher frequency of motion data would help password inference, since the raw data itself contains more information. As shown in the right three columns of Tab. 5, constantly sampling 200Hz motion data gives the best password inference accuracy: 66.3% for APL while 23.4% for PIN. However as discussed above, the energy consumption can be prohibitively high: it would drain on average 6.3% of battery per hour, which can easily halve the typical smartwatch lifetime [35]. On the other hand, we see that by using the adaptive motion sensing strategies, Snoopy is able to achieve comparable password inference accuracy: 64.5% for APL and 18.0% for PIN, while keep the energy consumption much lower. We also observe that the Snoopy front-end works better for APLs than PINs: the gaps comparing to 200Hz(const) are ~2% and ~5% respectively. This is because the adaptive sampling strategy tends to have a slight delay to switch to the high sampling rate mode when the user is starting to entering passwords, and thus could lose a small chunk of motion data at the beginning. For APLs it won't affect

inference much since the swiping event is continuous: one can use later data to smooth the sequence. For PINs, this is more sensitive since such data can often determine the first digit of the password and has an important impact on inference.

8.2 Possible Mitigations

Like most of the other side-channel attacks, Snoopy exploits the correspondence between the leaked motion data and the contents of entered passwords. Here we propose two possible types of countermeasures.

Context-aware Motion Sensor Access: One promising mitigation strategy is to enable context-aware motion sensor access, as proposed in [23]. The idea is to control the access level of motion sensors given the specific context, e.g. when users are typing on the touchscreen, it is better to limit motion sensor access, especially to those untrusted 3rd party applications. Depending on different context, one could consider control policies such as complete access blocking, reading order randomization and sampling rate reduction. However, the former two may have significant negative impact on some normal motion-dependent applications such as fitness apps, while the latter is more desirable in practice.

Eliminate Motion-Password Correspondence: The other possible countermeasure is to remove the one-to-one mapping between user generated motion data and password contents. For instance, we may insert noise to the motion data during password input, e.g. use random vibrations when the users tap or swipe, or randomly change the positions of the digits each time when users are required to authenticate, or generates different password keypads dynamically [31]. Or alternatively, one can establish more types of correspondence between the users and their passwords, e.g. using the particular tapping or swiping behaviour of a user to add another layer of authentication on top of standard PINs/APLs [45].

9 RELATED WORK

Attacking secrets via Side-channel: Leveraging physical sensors as a side to attack secrets has recently received lots of attention. The authors found that the MEMS gyro sensors are able to pick up low-frequency vibrations from ambient sounds. Aviv et al. demonstrated that it is possible to reconstruct a locking pattern by analyzing the oily residues left on the screen [4]. This method has limited application as oily residues can be altered by other on-screen activities after pattern drawing, and also requires an attacker to have physical access to the device. Li et al. proposed a keystroke inference framework using variations in WiFi signals. They observe that keystrokes on mobile devices will lead to different hand positions and finger motion which alter the channel properties, reflected in the channel state information (CSI). A similar idea is proposed in [1], where WiFi CSI is used to infer keystrokes on a physical keyboard. However, these classes of attacks require similar environments and are highly sensitive to nearby moving objects. Vision-based attacks are also well established. Shukla et al. [32] used video footage captured near the victim to decode the PIN entered on the smartphone. Ye et al. [43] recently extend [32] to APL and their method is robust to different lighting conditions. Though video-based side-channel attacks are very efficient in determining passwords, it is difficult for the attackers to access and locate video footage containing password input events. Compared with the above more direct attacks, eavesdropping motion sensor data is robust to environmental dynamics, is scalable, and can be achieved discreetly by a malicious app. On the hand, due to their motion tracking capability and pervasiveness, motion sensors are popular side channels for attackers. Gyrophone [27] presents a new type of threat to intercept human speech by using a smartphone gyroscope. Marquardt et al. demonstrated the *(Sp) iPhone*, and show that it is possible to use the accelerometer within an iPhone to recover text entered on a keyboard when the phone is placed nearby on a rigid surface [24].

Inferring Secrets on Smart Devices via Motion Sensors: Researchers have attempted to infer keystrokes on smart wearables via motion sensors [5, 7, 26, 28, 29, 42]. The core idea behind these works is similar to the aim of Snoopy: keystrokes on device screen lead to distinct force/attitude patterns. The motion data on smart

wearables can thus be used to infer entered secrets. TouchLogger [7, 8] and ACCessory [29] are early works, where ACCessory uses accelerometer only and TouchLogger utilizes both accelerometer and gyroscope to infer PINs. Similarly, TapLogger [42] refines previous techniques and uses a gyroscope to predict PIN-like secrets on smartphones. TapLogger uses a k-means clustering approach to extract the most likely classes (typically top 3). Given substantial observations of the secret (e.g. 32 PIN entry events), this is sufficient to estimate the true secret. Note that TapLogger uses manually extracted statistical features. TapPrints [28] advances the technique and extends inference capability beyond digits to English words. The papers mentioned above require accurate digit-wise classification, which is hard to achieve with smartwatch motion data (as shown in Sec. 6). In contrast, GestureLogger [5] infers keystrokes in an end-to-end manner rather than individually identifying each tapped digit. To this end, GestureLogger firstly designs a password database of 50 graphical passwords and 50 numerical PINs as possible passwords. It then develops a sequence classifier that infers the most likely match given the motion data sequence. However, GestureLogger uses handcrafted features for inference, which is not robust to the variability of scenarios [18]. For example, as demonstrated in experimental results (Sec. 6), the features designed for smartphones in GestureLogger did not work well in the context of smartwatches. Though redesigning new features for smartwatches is possible but the process needs domain knowledge (e.g., motion sensors). Unlike all the above, Snoopy is the only one using deep neural networks that is able to learn the best feature representations automatically through learning; its password inference framework can be easily applied to new scenarios without domain knowledge about the functioned sensors.

While TapPrints is specifically tailored to inferring PIN passwords, we provide a uniform approach that can be used to infer both PIN and APL (swiped) passwords. Even if one focuses on PINs only, we demonstrate a 2.5 fold improvement in accuracy compared to TapPrints. This is because TapPrints decouples the problems of segmentation and classification into two separate steps, whereas our approach handles them more robustly by tackling both tasks using the same Deep Neural Network architecture. When compared to previous work that has focused on APLs (GestureLogger), our work is fundamentally different as it can address not only APLs that exist in the training dataset, but also new previously unseen APLs. This is a significant benefit that increases the impact of the attack. As we can see in Tab. 6, Snoopy is the only approach requires little domain knowledge on attackers' side, which significantly lower the bar for attack deployment. In addition, whereas all prior art has focused on smartphones, we address the problem in the context of smartwatches. This is a far more challenging scenario, due to low SNR, in which previous approaches show very low performance compared to the proposed approach.

Smartwatch Security: As an increasingly ubiquitous device, the smartwatch has triggered new security issues. Wang et al. [40] and Liu et al. [20] pioneered this thread and have demonstrated the feasibility of inferring keystrokes on QWERTY keyboards by smartwatches. Their idea is that the keystroke-induced motions can be read from the motion sensors as long as smartwatches are worn on victims' wrists. Similar risks are also reported on the ATM machines, where victims' digital PINs are leaked through motion sensors on smartwatches. Maiti et al. [23] proposed a context-aware protection mechanism which identifies sensitive motion events (e.g., typing on the keyboard) and trigger sensor access controller accordingly. The protection mechanism has been proved to work effectively to mitigate smartwatch based side-channel attacks with least interruption for the third-party applications.

Though Snoopy is an attack framework based on smartwatch, the goals are fundamentally different. Above works use smartwatches as a side channel to infer secrets entered on external devices, while Snoopy thrives to infer the inputs entered into the watch through it screen.

Work	Participants Number	Domain Knowledge Required for Attackers		Password Extraction	PWD Type	Cross-user	Cross-device + Cross-user
		Digit/swipe Segmentation	Feature Engineering				
TouchLogger [7, 8]	1	✓	✓	✗	PIN	✗	✗
ACCessory [29]	4	✓	✓	✗	PIN	✗	✗
TapLogger [42]	3	✓	✓	✓	PIN	✗	✗
TapPrints [28]	10	✓	✓	✓	PIN	✗	✗
GestureLogger [5]	24	✓	✓	✗	APL PIN	✓	✗
Snoopy (proposed)	362	✗	✗	✓	APL PIN	✓	✓

Table 6. Comparison of related works and Snoopy in password inference.

10 CONCLUSION

In this paper, we studied the problem of password leaking on smartwatches via the on-board motion sensors. Although side-channel attacks based on motion data have been widely investigated on smartphones, the problem has so far been overlooked on smartwatch platforms. As a result, users are not fully aware of the risks and potential consequences. To our knowledge, this is the first work that demonstrates the feasibility of attacking passwords (PIN and Android Pattern Lock) on smartwatches using motion sensors. The proposed Snoopy system can disguise itself as a normal app (for fitness or wellbeing monitoring), and can successfully eavesdrop motion data in the background while passwords are entered. The extracted motion data is uploaded to the cloud, where Snoopy infers the contents of passwords using deep neural networks trained with crowd-sourced data. We collected large scale datasets (3 different sites, 362 users and >50k password entires), and compared the performance of Snoopy with state-of-the-art methods proposed for smartphones. Our findings are: a) Snoopy can accurately extract the password-positive segments of motion data (up to 98% accuracy) in real-time on smartwatches, without consuming significant power/computational resources; b) although it is more challenging to infer passwords on smartwatches due to the much lower signal-to-noise ratio, Snoopy is able to achieve up to 95% success rates when attacking Android Pattern Lock within 10 attempts and 86% for PIN numbers (the best competing approaches achieving 23% and 30% respectively); c) Snoopy is robust to user and device heterogeneity. Compared with the best competing approach, it offersoffers a $\sim$ 3-fold improvement in the accuracy of inferring passwords. d) in practice it is vital to select the appropriate deep network architecture and regularisation parameters for Snoopy to work well. In summary, through the use of state-of-the-art deep learning, we have presented a universal technique that works for both PINs and pattern locks, without requiring any hand-engineering of features. By lowering the barrier to attackers in terms of engineering effort, the likelihood of being able to successfully compromise smart devices becomes significantly higher, simply by harvesting innocuous motion data from victims. In light of these risks, we recommend that motion sensors should be regarded as a carrying a far higher security risk than they currently are, and should have adequate countermeasures in place or require increased permissions.

REFERENCES

[1] Kamran Ali, Alex X Liu, Wei Wang, and Muhammad Shahzad. 2015. Keystroke recognition using wifi signals. In *ACM Conference on Mobile Computing and Networking, MobiCom*.

[2] Ltd Alipay.com Co. 2017. Alipay - Makes Life Easy. https://itunes.apple.com/us/app/alipay-makes-life-easy/id333206289?mt=8. (2017).

[3] Daniel Amitay. 2014. Most Common iPhone Passcodes. http://danielamitay.com/blog/2011/6/13/most-common-iphone-passcodes. (2014).

[4] Adam J Aviv, Katherine L Gibson, Evan Mossop, Matt Blaze, and Jonathan M Smith. 2010. Smudge Attacks on Smartphone Touch Screens. *USENIX Workshop on Offensive Technologies, Woot* (2010).

[5] Adam J Aviv, Benjamin Sapp, Matt Blaze, and Jonathan M Smith. 2012. Practicality of accelerometer side channels on smartphones. In *Proceedings of the 28th Annual Computer Security Applications Conference, ACSAC*.

[6] Baidu. 2016. Offline Alipay Setup. http://jingyan.baidu.com/article/ce4366492b02263773afd3f0.html. (2016).

[7] Liang Cai and Hao Chen. 2011. TouchLogger: Inferring Keystrokes on Touch Screen from Smartphone Motion.. In *Proceedings of the 6th USENIX Conference on Hot Topics in Security, HotSec*. USENIX.

[8] Liang Cai and Hao Chen. 2012. On the practicality of motion based keystroke inference attack. In *International Conference on Trust and Trustworthy Computing, TRUST*.

[9] François Chollet et al. 2015. Keras: Deep learning library for theano and tensorflow. *URL: https://keras. io/k* (2015).

[10] John Duchi, Elad Hazan, and Yoram Singer. 2011. Adaptive subgradient methods for online learning and stochastic optimization. *Journal of Machine Learning Research* 12, Jul (2011), 2121–2159.

[11] Thomas Graziani. 2016. smartwatch report. https://walkthechat.com/apple-watch-wechat-are-adding-further-integrated-features/. (2016).

[12] Marian Harbach, Alexander De Luca, and Serge Egelman. 2016. The anatomy of smartphone unlocking: A field study of android lock screens. In *ACM Conference on Human Factors in Computing Systems, CHI*.

[13] Karl Moritz Hermann, Tomas Kocisky, Edward Grefenstette, Lasse Espeholt, Will Kay, Mustafa Suleyman, and Phil Blunsom. 2015. Teaching machines to read and comprehend. In *Advances in Neural Information Processing Systems, NIPS*.

[14] Sepp Hochreiter, Yoshua Bengio, Paolo Frasconi, and Jürgen Schmidhuber. 2001. Gradient flow in recurrent nets: the difficulty of learning long-term dependencies. (2001).

[15] Sepp Hochreiter and Jürgen Schmidhuber. 1997. Long short-term memory. 9, 8 (1997), 1735–1780.

[16] Diederik Kingma and Jimmy Ba. 2014. Adam: A method for stochastic optimization. In *International Conference on Learning Representations, ICLR*.

[17] David Krueger, Tegan Maharaj, János Kramár, Mohammad Pezeshki, Nicolas Ballas, Nan Rosemary Ke, Anirudh Goyal, Yoshua Bengio, Hugo Larochelle, Aaron Courville, et al. 2017. Zoneout: Regularizing rnns by randomly preserving hidden activations. (2017).

[18] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. 2015. Deep learning. *Nature* 521, 7553 (2015), 436–444.

[19] Mengyuan Li, Yan Meng, Junyi Liu, Haojin Zhu, Xiaohui Liang, Yao Liu, and Na Ruan. 2016. When CSI Meets Public WiFi: Inferring Your Mobile Phone Password via WiFi Signals. In *ACM SIGSAC Conference on Computer and Communications Security, CCS*.

[20] Cihang Liu, Lan Zhang, Zongqian Liu, Kebin Liu, Xiangyang Li, and Yunhao Liu. 2016. Lasagna: towards deep hierarchical understanding and searching over mobile sensing data. In *ACM Conference on Mobile Computing and Networking, MobiCom*.

[21] Xiangyu Liu, Zhe Zhou, Wenrui Diao, Zhou Li, and Kehuan Zhang. 2015. When good becomes evil: Keystroke inference with smartwatch. In *CCS*. ACM.

[22] Marte Dybevik Løge. 2015. *Tell Me Who You Are and I Will Tell You Your Unlock Pattern.* Master's thesis. NTNU.

[23] Anindya Maiti, Oscar Armbruster, Murtuza Jadliwala, and Jibo He. 2016. Smartwatch-based keystroke inference attacks and context-aware protection mechanisms. In *CM on Asia Conference on Computer and Communications Security AsiaCCS*.

[24] Philip Marquardt, Arunabh Verma, Henry Carter, and Patrick Traynor. 2011. (sp) iphone: Decoding vibrations from nearby keyboards using mobile phone accelerometers. In *ACM SIGSAC Conference on Computer and Communications Security, CCS*.

[25] James L Massey. 1994. Guessing and entropy. In *IEEE International Symposium on Information Theory (ISIT)*.

[26] Maryam Mehrnezhad, Ehsan Toreini, Siamak F. Shahandashti, and Feng Hao. 2017. Stealing PINs via mobile sensors: actual risk versus user perception. *International Journal of Information Security* (2017).

[27] Yan Michalevsky, Dan Boneh, and Gabi Nakibly. 2014. Gyrophone: Recognizing Speech from Gyroscope Signals.. In *USENIX Security*.

[28] Emiliano Miluzzo, Alexander Varshavsky, Suhrid Balakrishnan, and Romit Roy Choudhury. 2012. Tapprints: your finger taps have fingerprints. In *International Conference on Mobile Systems, Applications, and Services, MobiSys*. ACM.

[29] Emmanuel Owusu, Jun Han, Sauvik Das, Adrian Perrig, and Joy Zhang. 2012. ACCessory: password inference using accelerometers on smartphones. In *Proceedings of the Twelfth Workshop on Mobile Computing Systems & Applications, HotMobile*.

[30] Allied Market Research. 2016. Smartwatch Market is Expected to Reach 32.9 Billion, Globally, by 2020. https://goo.gl/DJjHYs. (2016).

[31] Muhammad Shahzad, Alex X Liu, and Arjmand Samuel. 2013. Secure unlocking of mobile touch screen devices by simple gestures: you can see it but you can not do it. In *Proceedings of the 19th annual international conference on Mobile computing & networking, (MobiCom)*.

[32] Diksha Shukla, Rajesh Kumar, Abdul Serwadda, and Vir V Phoha. 2014. Beware, your hands reveal your secrets!. In *ACM SIGSAC Conference on Computer and Communications Security, CCS*.
[33] Elizabeth Stobert and Robert Biddle. 2014. The password life cycle: user behaviour in managing passwords. In *USENIX Symposium On Usable Privacy and Security, SOUPS*.
[34] Ilya Sutskever, Oriol Vinyals, and Quoc V Le. 2014. Sequence to sequence learning with neural networks. In *Advances in neural information processing systems, NIPS*.
[35] TechRadar. 2015. Sony Smartwatch 3 review. http://www.techradar.com/reviews/sony-smartwatch-3. (2015).
[36] Tijmen Tieleman and Geoffrey Hinton. 2012. Lecture 6.5-rmsprop: Divide the gradient by a running average of its recent magnitude. *COURSERA: Neural networks for machine learning* (2012).
[37] Colton J Turner, Barbara S Chaparro, and Jibo He. 2017. Text input on a smartwatch QWERTY keyboard: tap vs. trace. *International Journal of Human–Computer Interaction* (2017).
[38] Sebastian Uellenbeck, Markus Dürmuth, Christopher Wolf, and Thorsten Holz. 2013. Quantifying the security of graphical passwords: the case of android unlock patterns. In *ACM SIGSAC conference on Computer & communications security, CCS*.
[39] Emanuel Von Zezschwitz, Paul Dunphy, and Alexander De Luca. 2013. Patterns in the wild: a field study of the usability of pattern and pin-based authentication on mobile devices. In *ACM Conference on Human Factors in Computing Systems, CHI*.
[40] He Wang, Ted Tsung-Te Lai, and Romit Roy Choudhury. 2015. Mole: Motion leaks through smartwatch sensors. In *MobiCom*. ACM.
[41] Zhuoling Xiao, Hongkai Wen, Andrew Markham, and Niki Trigoni. 2014. Lightweight map matching for indoor localisation using conditional random fields. In *International Conference on Information Processing in Sensor Networks, IPSN*. IEEE.
[42] Zhi Xu, Kun Bai, and Sencun Zhu. 2012. Taplogger: Inferring user inputs on smartphone touchscreens using on-board motion sensors. In *Proceedings of the 5th ACM conference on Security and Privacy in Wireless and Mobile Networks*.
[43] Guixin Ye, Zhanyong Tang, Dingyi Fang, Xiaojiang Chen, Kwang In Kim, Ben Taylor, and Zheng Wang. 2017. Cracking Android pattern lock in five attempts. (2017).
[44] Wojciech Zaremba, Ilya Sutskever, and Oriol Vinyals. 2014. Recurrent neural network regularization. (2014).
[45] Nan Zheng, Kun Bai, Hai Huang, and Haining Wang. 2014. You are how you touch: User verification on smartphones via tapping behaviors. In *IEEE 22nd International Conference on Network Protocols (ICNP)*.

APPENDIX

Recurrent Neural Networks (RNNs) based Sequence Learning

In our context, the task of password inference is essentially learning a function between the password related motion data to the finite labels in the password database P. The motion data is inherently a time series, which captures the continuous posture changes of the smartwatch induced by user tapping or swiping. In addition, our problem is more challenging than standard sequence modelling since a) the input length can be variable, e.g. APLs can have different lengths, and b) the temporal correlations within data are strong, e.g. the likelihood of tapping on or swiping to a particular position depends very much on previous taps/swipes. Therefore, in this paper we use RNNs to model the motion data, which can take arbitrary length of input, and return the most likely password as output. In the following we explain how the standard RNN architecture solves this type of sequence learning problem. In the last two subsections we describe how to extend the standard architecture to cope with our particular password inference problems.

Basic RNN Architecture: Concretely, at each timestamp $\mathbf{k}$ a standard RNN keeps an internal hidden state $\mathbf{h}_k$ to describe the temporal dependencies, and given an input $\mathbf{x}_k$, the RNN updates its state by:

$$\begin{aligned} \mathbf{h}_k &= \mathcal{H}(\mathbf{W}_{xh}\mathbf{x}_k + \mathbf{W}_{hh}\mathbf{h}_{k-1} + \mathbf{b}_h) \\ \mathbf{u}_k &= \mathbf{W}_{hu}\mathbf{h}_k + \mathbf{b}_u \end{aligned} \tag{6}$$

where $\mathbf{W}_{xh}$, $\mathbf{W}_{hh}$ are the weights of the current input $\mathbf{x}_k$ and previous state $\mathbf{h}_{k-1}$, and $\mathbf{b}_h$ is the bias vector. $\mathcal{H}$ is an element-wise non-linear activation function, e.g., sigmoid or hyperbolic tangent function. The network output $\mathbf{u}_k$ is evaluated as a linear combination of the updated hidden state $\mathbf{h}_k$ and a bias vector $\mathbf{b}_u$.

In practice, $\mathbf{u}_k$ may appear at multiple timestamp, or only exist at a single (in most cases the last) timestamp. The former type of network is able to map the input sequence to an output sequence, which is particularly useful in scenarios such as machine translation. The latter type generates a single output, for example a label based on

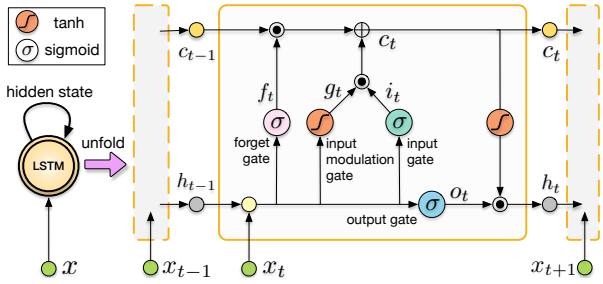

Fig. 17. Folded and unfolded LSTMs and the internal structures of its unit. $\odot$ and $\oplus$ denote element-wise product and addition of two vectors, respectively.

the input sequence, and thus is often considered in applications such as text sentimental analysis. As discussed above, the password inference problem studied in this paper is to compute the most likely password (i.e. label) within the database P given the sequence of motion data, which falls naturally into the latter category. Therefore in the following text, we only focus on those RNNs with a single output node, as shown in Fig.5 (Left).

RNNs with LSTMs: Although in theory the basic RNN is able to model sequences with arbitrary length, in practice it often suffers from the gradient vanishing and exploding problems [14]. That is, it cannot capture the long-term dependencies well when the length of input sequences becomes large. Therefore for long input sequences, RNNs with Long Short Term Memory units (LSTMs) [15] are often considered. Essentially, LSTMs use self recurrent units (*memory cells*) to explicitly determine which previous hidden states to "remember" or "forget", and thus provides a memory mechanism to capture dependencies spanning over many timesteps in the past. Fig. 17 shows an example of the LSTM units, in which the gates (forget/input/output) control the flow of information. Formally, the LSTM updates its own cell state $\mathbf{c}_k$ and the hidden state $\mathbf{h_k}$ as follows:

$$
\begin{aligned}
\mathbf{i}_k &= \sigma(\mathbf{W}_{xi}\mathbf{x}_k + \mathbf{W}_{hi}\mathbf{h}_{k-1} + \mathbf{b}_i) \\
\mathbf{g}_k &= \tanh(\mathbf{W}_{xg}\mathbf{x}_k + \mathbf{W}_{hg}\mathbf{h}_{k-1} + \mathbf{b}_g) \\
\mathbf{f}_k &= \sigma(\mathbf{W}_{xf}\mathbf{x}_k + \mathbf{W}_{hf}\mathbf{h}_{k-1} + \mathbf{b}_f) \\
\mathbf{c}_k &= \mathbf{f}_k \odot \mathbf{c}_{k-1} + \mathbf{i}_k \odot \mathbf{g}_k \\
\mathbf{o}_k &= \sigma(\mathbf{W}_{xo}\mathbf{x}_k + \mathbf{W}_{ho}\mathbf{h}_{k-1} + \mathbf{b}_o) \\
\mathbf{h}_k &= \mathbf{o}_k \odot \tanh(\mathbf{c}_k)
\end{aligned}
\tag{7}
$$

where $\mathbf{i}_k$ and $\mathbf{g}_k$ are the input gates, i.e. govern which information in the previous hidden state $\mathbf{h_{k-1}}$ to remember, while $\mathbf{f}_k$ is the forget gate that controls which part in $\mathbf{h_{k-1}}$ to forget, given the current input $\mathbf{x}_k$. The cell state $\mathbf{c}_k$ of this LSTM unit is then updated by combining information from those gates. $\mathbf{o}_k$ is the output gate, which is then fused with the current cell state $\mathbf{c}_k$ to update the hidden state $\mathbf{h_k}$. Note that σ is the sigmoid function, $\tanh$ is the hyperbolic tangent function, and $\odot$ is the element-wise product between vectors.

Cost Function and Optimisation: Given an input sequence $\mathbf{X} = (x_1, x_2, \cdots, x_T)$ with length of T (in our case the 6 axis motion data), the above RNN essentially computes the likelihood of the labels $y \in \{1, 2, \cdots, M\}$:

$$
\begin{aligned}
p(y|\mathbf{X}) &= p(y|x_1, x_2, \cdots, x_T) \\
&= softmax(\mathbf{W}_{hu}\mathbf{h}_T + \mathbf{b}_u)
\end{aligned}
\tag{8}
$$

where we assume there is M possible labels in total. If the true label y is known for the input sequence $\mathbf{X}$, then training the network is equivalent to finding the optimal parameters (weights) θ where:

$$\theta^* = \underset{\theta}{\text{argmax}} \quad p(y|\mathbf{X};\theta) \tag{9}$$

In practice, we typically use the cross entropy errors between the predicted and true labels as the cost function, which is defined as:

$$\mathcal{L}(\mathbf{X}, y) = \sum_{j=1}^{M} \mathbf{1}\{y = j\} \log p(\widehat{y}_j) \tag{10}$$

where $p(\widehat{y}_j) \in [0, 1]$ is the probability of label j predicted by the network, and $\mathbf{1}\{y = j\}$ is an indicator function which returns 1 if the true label y is j. To optimize this cost function, there are various gradient-based techniques such as AdaGrad [10] and RMSProp [36]. In this paper we use Adam [16], which combines the advantage of the two and is very efficient for training deep RNNs with LSTMs.

Regularization: Like other deep neuronal networks, the above RNNs with LSTMs can get overfit quickly. This is because by design the input of the network can have arbitrary length while the dimension of the output is fixed. Therefore through training, the RNNs tend to learn an input-dependent transition operator, which governs how to "fold" the variable length input sequences into the hidden states and "squeeze" them into fixed output vectors. However, as the training set won't cover sequences with all possible lengths, when applying the learned transition operator to an unseen test sequence, the dynamics of RNNs can be very sensitive to minor perturbations in the hidden states at different timestamps [17]. Therefore to make the RNNs practically useful, we need good regularization techniques to balance the fitness of training and generalization capability. In this paper, we consider a dropout technique inspired by [44], which randomly shuts down a subset of the network by disabling the corresponding feed-forward connections during training. Therefore, the dropout technique deliberately corrupts the information maintained by the memory units at some timestamps, but not all of them. In this way, we force the network to learn the general knowledge but not just specific features in the training data, and thus make it more robust in practice.

Received May 2017; revised August 2017 and September 2017; accepted October 2017