PROJECT SANKHYA

KEYSTROKE INFERENCE USING SIDE CHANNEL ATTACK ON SMART PHONE SENSORS

EE5180- Introduction to Machine Learning

Group-14

EE18B026 – P S Sainath,

EE18B030 – Reshma Rukshinda,

EE18B046 – D Sumanth,

EE18B060 – P Arun Kumar Reddy

Problem Statement

Due to the advancement in technology and excessive usability of smartphones in various domains (e.g., mobile banking etc), smartphones became more prone to malicious attacks. We examine the problem of keystroke inference from the side-channel attack of sensor data.

The objective of our project is to build a machine learning model that predicts the Key pressed using sensor data.

Background and Motivation

Security and privacy are very important in this age of technological significance. We present this attack as a form of showing the vulnerability in the system which may lead to fixing this in the future.

In terms of human-computer interaction, the keyboard is the key input device used to input data and commands in smartphones. Keyboards are commonly used to enter personal data such as PIN codes, passwords, and credit card information along with the usual text (such as text messages, emails, etc.). **Accelerometer and gyroscope readings can be easily accessed using:**

- JavaScript from browsers
- Malicious apps running in background

We observe that by using these sensor readings a learning model can easily be trained to accurately determine soft keyboard inputs on an Android device. **This usage greatly reduces the trials required in cracking the passcode or PIN.**

Dataset

As there aren't any readily available datasets, we developed an application named SANKHYA to collect data. The SANKHYA app allows the user to input the number prompt on the screen. It also collects information regarding hand configuration (Users have to select the hand configuration which he/she uses to type i.e 1 hand, 2 hands, forefinger).

```
Data File:04/16/2021 10:57:54 PM

16-04-2021 10-57-43-PM
Reshma 20M H1
{"is_magnet":true,"is_alpha":false,"sampling_rate":"100"}
1967-9
2164-3
3440-0
3622-4
4620-1
4905-0
6017-8
18;-1.21556;6.58984;7.48834;9.65722;16.84038;-0.7451;35.52856;-17.89703;-0.2243
41;-1.14377;6.52762;7.58646;8.61789;16.49277;-1.73622;35.52856;-17.89703;-0.2243
63;-1.1689;6.42354;7.71327;7.14159;15.0348;-2.87507;35.70709;-18.06488;-1.02234
79;-1.1689;6.42354;7.71327;5.18875;13.30344;-3.47501;35.70709;-18.06488;-1.02234
95;-1.0361;6.42712;7.47757;2.44073;11.80196;-3.29649;35.70709;-18.06488;-1.02234
110;-0.72025;6.35893;7.37827;1.25349;10.80261;-3.36754;34.89685;-18.33344;-2.87781
124;-0.41516;6.24168;7.54338;0.47091;9.56083;-3.60484;34.89685;-18.33344;-2.87781
145;-0.34697;6.11964;7.45604;-0.24335;6.73381;-4.04755;34.89685;-18.33344;-2.87781
165;-0.49173;6.03949;7.38545;-0.35899;2.54018;-5.32975;35.3775;-18.31512;-3.01361
181;-0.49173;6.03949;7.38545;-0.49203;-3.67308;-4.56723;35.3775;-18.31512;-3.01361
197;-0.27876;6.01198;7.69293;-1.72653;-11.36687;0.16245;35.3775;-18.31512;-3.01361
245;0.18784;5.88754;8.20859;-7.05165;-17.30446;6.89335;34.65729;-18.56995;-5.26886
261;0.18904;6.01436;8.17628;-9.05194;-18.90034;6.05737;34.94873;-18.85681;-5.13611
278;0.00121;6.10889;8.19302;-12.81367;-21.75999;6.67427;34.94873;-18.85681;-5.13611
294;-0.04787;6.13519;8.08057;-15.55393;-22.95182;8.2673;34.94873;-18.85681;-5.13611
312;0.05743;6.14955;7.84248;-19.04986;-23.34015;10.47075;34.65576;-19.15588;-5.26581
325;0.05743;6.14955;7.84248;-18.71338;-22.27629;10.59086;34.65576;-19.15588;-5.26581
351;-0.25842;6.10051;7.5374;-14.63684;-19.65063;9.96227;34.65576;-19.15588;-5.26581
370;-0.35773;6.12204;7.5338;-11.38481;-17.54862;8.48452;34.4635;-18.86902;-5.26581
384;-0.35773;6.12204;7.5338;-9.4926;-16.13627;7.36111;34.4635;-18.86902;-5.26581
404;-0.52045;5.80618;7.84128;-5.15366;-13.3261;4.637;34.4635;-18.86902;-5.26581
418;-0.87338;5.4377;8.28516;-1.68508;-11.13709;2.32385;34.56268;-19.06433;-5.13306
431;-0.6329;6.04906;8.26601;-0.86521;-9.35505;1.22555;34.56268;-19.06433;-5.13306
445;-0.6329;6.04906;8.26601;-0.45776;-6.22067;0.48202;34.56268;-19.06433;-5.13306
459;-0.37688;6.37569;8.0363;-0.11888;-1.61544;0.12518;34.96704;-19.66553;-4.06799
477;0.05983;6.53481;7.96332;-0.33786;-0.01046;-0.56523;34.96704;-19.66553;-4.06799
491;0.34337;6.65685;7.54219;-1.91005;3.17522;-0.78415;34.96704;-19.66553;-4.06799
509;-0.11366;6.71187;7.12822;-2.72958;4.84817;0.94195;34.1095;-20.00122;-3.39508
537;-0.47379;6.58026;6.87338;-0.73548;5.51462;2.18259;34.1095;-20.00122;-3.39508
554;-0.3649;6.5025;6.5719;3.5361;6.21176;2.43838;34.1095;-20.00122;-3.39508
573;0.07419;6.54677;6.28595;8.14044;5.52414;2.30332;34.58252;-20.56274;-3.79486
```

Figure 1: Raw data collected from the app SANKHYA

As the user starts pressing the key, the app records the keystrokes and sensor data in parallel.

- Accelerometer (in X, Y, Z axes)
- Gyroscope (in X, Y, Z axes)
- Magnetometer (in X, Y, Z axes)

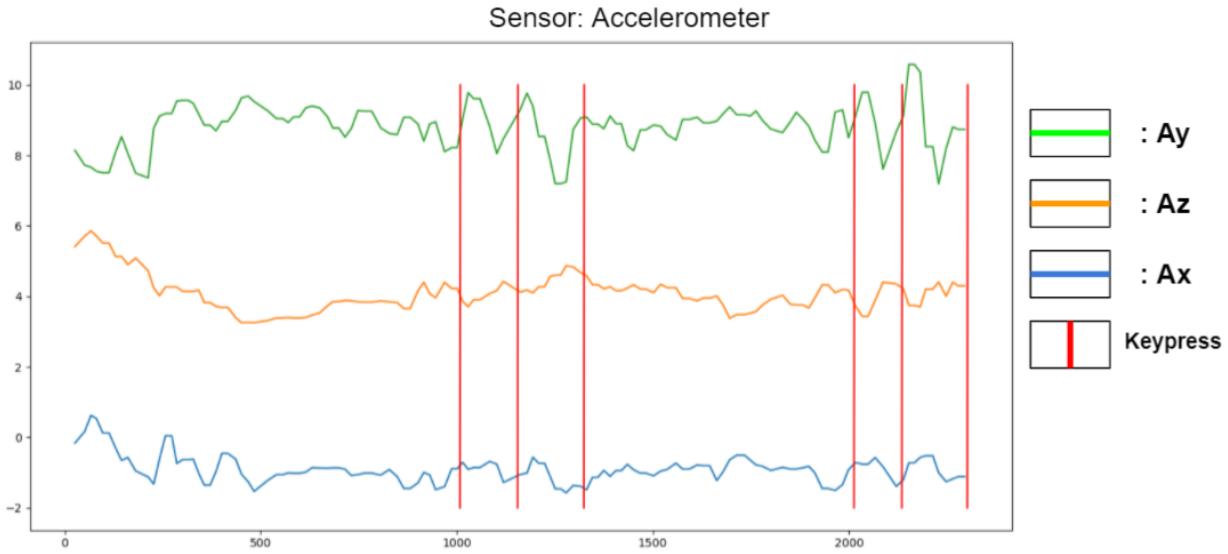

Figure 2: Plot of accelerometer sensor reading

(after feature extraction discussed next)

Analyzing the Data

1. Feature extraction
    - Raw data obtained from the app consists of information related to the user, hand configuration, timestamps of keypress and sensor data recorded for a period of time.
    - Sensor readings are collected at a frequency of 100 Hz. Using the time stamps we extract the sensor readings related to a particular keypress from the sensor data.
    - We label the data extracted at the timestamp with that particular key.

2. Pre-processing the Data:
    - **Effect of gravity**: Due to the inclination present in the natural holding position of the phone, the gravity has two components: gy and gz which add to the accelerometer readings. We remove this extra component by averaging the sensor data and subtracting the average from the actual data.
    - **Effect of base orientation**: This is analogous to gravity effect but the magnetic version of it (in 3 dimensions). We use the same technique to remove these components from the sensor data.
    - **Backspace problem:**
        - Backspace is used to correct the mistakenly typed key.
        - We cannot say whether a backspace is pressed or not just by seeing text. Thus 'Backspace' cannot be visualised with text whereas it's a keypress in the physical sense.
        - Ignoring backspace can mislead the model to predict the sensor data related to 'backspace' as any key between 0-9.

      We tackled this problem by writing a special algorithm that registers 'backspace' as a special key. We label the sensor data related to 'backspace' using the symbol '`'(backtick). Thus, our model can be trained to predict 'backspace' too.

Input Methods:

Single time-instant values input:

- We will extract the sensor values exactly at keypresses and use them as input

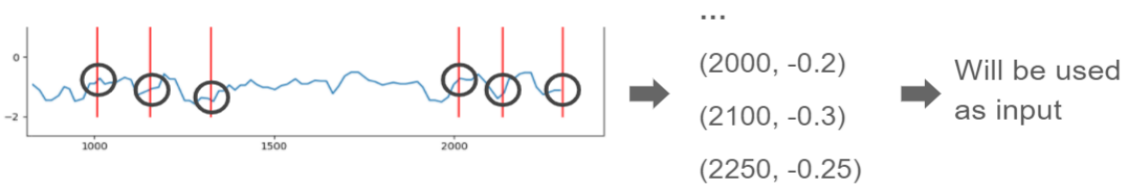

Figure 3

Sequence input:

- We give the whole sequence* as input to the model (especially to the sequential models).

*whole sequence is split into smaller sequences for each key press and they are used.

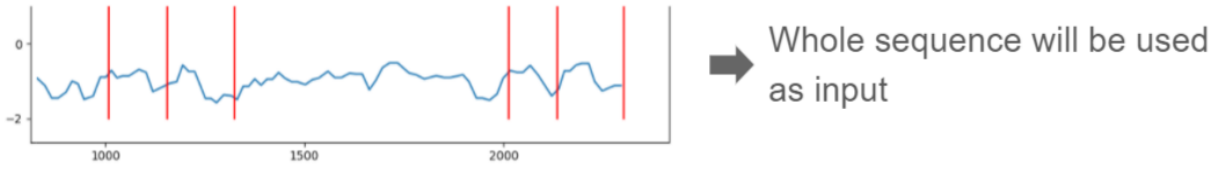

Figure 4

Machine learning Models

We cannot comment on the nature of sensor data (i.e linearly separable etc). So, we plan to work with different algorithms and see how well they work.

As this is a classification problem, we choose following models:

- Multinomial Logistic regression
- Support Vector Machine
- Multi-Layer Perceptron
- Convolutional Neural Networks

1. **Multinomial Logistic regression:**

   We are dealing with small data sets, so we chose the 'liblinear' optimizer. We also tried working with other optimizer and penalty functions. There isn't much change in the accuracy obtained, leaving 'liblinear' to be better classified.

2. **Support Vector Machine:**

   SVM comes handy when dealing with non-linear data. We use the kernel trick to see how well it classifies.

   - Linear and rbf(with low value of gamma) kernels gave almost same accuracy
   - Whereas for poly with a degree 2 and C=1000, it's performance is similar to Linear with C=1. Performance of poly decreases with increase in value of degree and decrease in C value.

3. **Multi-Layer Perceptron:**

   MLP has an ability to deal with non- linearly separable data. We use different activation functions like logistic, identity, relu and different solver functions(optimizers) - 'lbfgs', 'adam'.

We observed better accuracy with the following parameters as shown below.

```
from sklearn.neural_network import MLPClassifier
clf = MLPClassifier(solver='adam',activation='relu', hidden_layer_sizes=(22, 20), random_state=1,max_iter = 400)
clf.fit(x_train, y_train)
```

```
#y_predict = clf.predict(x_test)
clf_score = clf.score(x_test,y_test)
print(clf_score)
#store_result(clf, clf_score, 'pre-processed')
```

```
0.31886155330438976
```

4. **Convolutional Neural Networks**:

We used Single time-instant value input in Logistic Regression, SVM and MLP. Here we use sequence input. Cubic spline is used to build a continuous sequence of data points and thus a new uniformly spaced sequence is extracted and fed to CNN model.

Performance of this model is very low. We expect this happens due to the data which might not be precise for different keys and when we try to consider a small part of the sequence, it also gets affected by other values.

Evaluation

Apart from using MSE, accuracy as evaluation metrics, we also used a **coordinate system** to depict how close our models predict for a given keypress. The coordinate system consists of 10 digits (0-9) and other keys as shown in the figure. In order to show how close our model predicts the data, we plotted **probability heat maps**. A picture of the heat map is shown below.

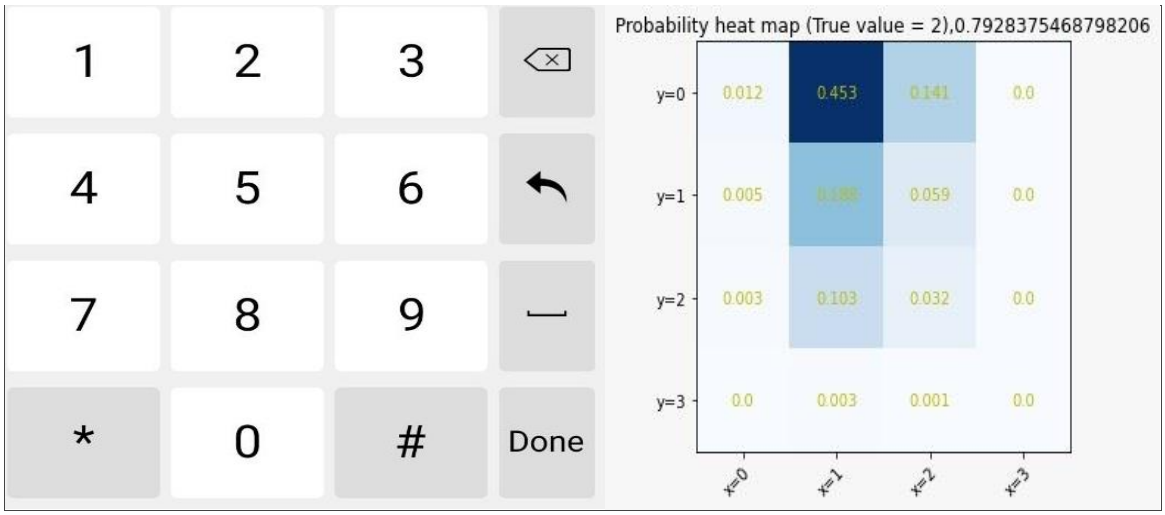

Figure 5

The heat map here shows the probabilities of the keys being predicted where actually key '2' is pressed. The keys next to 2(3,5) show significant probability and the probability decreases as the distance between the keys increases.

We defined a metric called **custom error** (CE). Custom error is 0 when the key is predicted with probability 1, and 4 when the key is predicted with probability 0. In the above heat map, custom error is 0.79. Another metric called **custom accuracy** (CA) is defined as 1-CE/4. Custom accuracy is 1 for correct predictions with probability 1, and 0 for predictions with probability 0. In the above example, custom accuracy is 0.8.

Results

- Accuracy obtained using the above -mentioned model varies from 25% - 32%.
- Performance varies depending on the data set used (Single person, combination datasets generated by different users). Whereas we obtain a higher accuracy using a dataset of a single person.

Observations

We can observe that in the case of logistic regression and SVM, accuracy of the above models is higher when a linear model is used compared to poly with higher degree. Though the accuracy is not very high, we can expect our data to be more on the linear side compared to poly. We also observe increase in accuracy when the user press a particular key in a similar fashion every time.

Improvements

- Range of data related to different keys is shown in the following figure. (x axis reading of accelerometer)

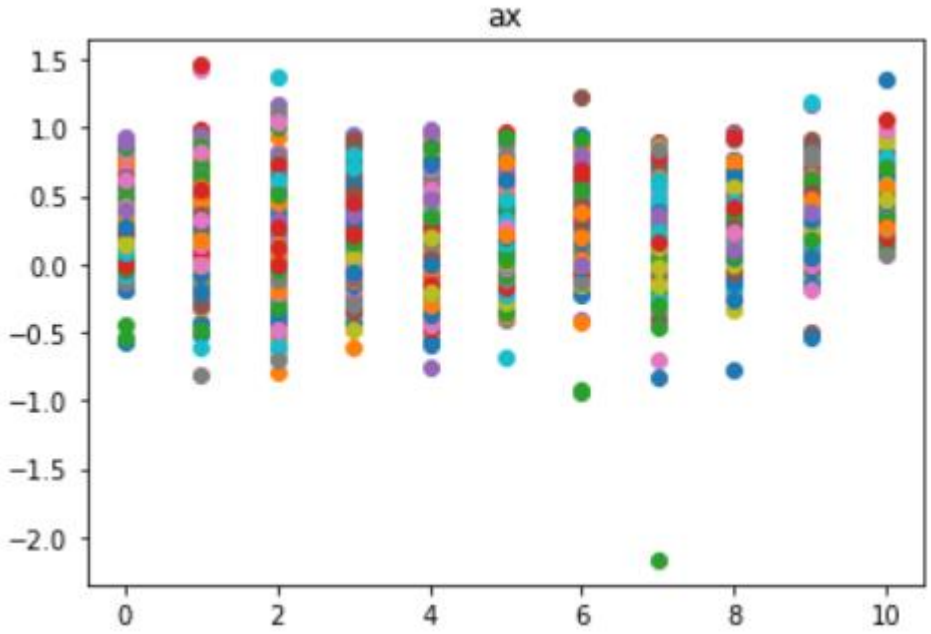

Figure 6

- We can see that there isn't much difference in the range of data of different keys. This can be a reason for models to not classify the data accurately.
- Thus, by improving the app SANKHYA to collect the data more precisely can increase the performance.
- And developing the app to collect all the samples at a single frequency can also help improve the results.

References:

https://web.cs.ucdavis.edu/~hchen/paper/trust2012touch.pdf
http://www.hotmobile.org/2012/papers/HotMobile12-final42.pdf
https://dl.acm.org/doi/10.1145/2185448.2185465
https://dl.acm.org/doi/10.1145/2766498.2766511

✦ END✦