# SSG: Sensor Security Guard for Android Smartphones[⋆]

Bodong Li, Yuanyuan Zhang, Chen Lyu, Juanru Li, and Dawu Gu

Lab of Cryptology and Computer Security,
Shanghai Jiao Tong University, Shanghai, China
{uchihal,yyjess,chen_lv,jarod,dwgu}@sjtu.edu.cn
WWW home page:http://loccs.sjtu.edu.cn/wiki/doku.php

**Abstract.** The smartphone sensors provide extraordinary user experience in various Android apps, e.g. sport apps, gravity sensing games. Recent works have been proposed to launch powerful sensor-based attacks such as location tracing and sound eavesdropping. The use of sensors does not require any permission in Android apps, so these attacks are very difficult to be noticed by the app users. Furthermore, the combination of various kinds of sensors generates numerous types of attacks which are hard to be systematically studied.
To better address the attacks, we have developed a taxonomy on sensor-based attacks from five aspects. In this work, we propose a sensor API hooking and information filtering framework, *Sensor Security Guard* (SSG). Unlike any rough hooking framework, this system provides fine-grained processing for different security levels set by the users, or by default. The sensor data is blocked, forged or processed under different mode strategies and then returned to the apps. In addition, according to the taxonomy, SSG develops fine-grained corresponding countermeasures. We evaluate the usability of SSG on 30 popular apps chosen from Google Market. SSG does not cause any crash of either the Android system or the apps while working. The result indicated that SSG could significantly preserve the users' privacy with acceptable energy lost.

**Key words:** Hook, Sensor API, Android, Security

## 1 Introduction

In recent years, the popularity of the Android smartphones provides extraordinary user experience with the assist of various built-in sensors on the devices that are able to measure various motion, orientation, and ambient conditions. Numerous Android apps are utilizing sensors nowadays, such as games, IMs and sport-related apps. For example, a three-dimensional accelerometer commonly known as a motion sensor, is now deployed in Nike+ running application [2] to record and calculate the running distance.

[⋆] Major program of Shanghai Science and Technology Commission (Grant No: 15511103002): Research on Mobile Smart Device Application Security Testing and Evaluating

When users are enjoying these sensor-aided apps, the information collected from the sensors might be revealing the privacy secretly at mean time. A few embedded hardware including GPS sensor, microphone and camera have attracted most interests in privacy protection research, for they provide users' precise physical location, voice or photos, straightforwardly. But the privacy leaked from the integrated sensors such as orientation sensors, magnetometers, accelerometers, etc, are barely noticed nor studied. For example, the information gathered by accelerometer is the smartphone's acceleration at a point in time. It seems irrelevant to user privacy, however, an attacker can derive the possible moving direction on top of it [9]. Even worse, the Android apps do not require any permissions to use such motion sensors (accelerometer and gyroscope).

Such stealthy use of the sensors might cheat most users that the sensors are quite safe. However, it has brought five grave privacy leak issues on identity theft, location tracing, password eavesdropping, etc. For example, with long-term sensor data collecting and analysis, users' identities could be inferred from gait patterns [11, 15]. By gathering sensitive information, a malicious app could also disclose personal location tracing [9, 13, 10], past speeches [12, 6] and inputs to keyboard [5, 3, 14]. Furthermore, an attacker is able to identify a user's mobile device by measuring anomalies of sensors [7, 4, 8].

To better address the security issues, we first classify the attacks that are forged on top of the sensors into five categories by the compromised resource/privacy. Previous works on Android sensor information abuse are still interested in privacy related information as location, password, user identity, etc. Accordingly, we propose five threat categories, 1) location tracing, 2) sound eavesdropping, 3) keystroke monitoring, 4) device fingerprint distinguishing, and 5) user identity pinpointing. The characteristics of the threats are discrete, and the exploit methods vary from each other. Usually, most attacks exploit more than one sensors, as conspiracy attacks.

In order to design an all-in-one solution to protect privacy from sensor-based attacks, we propose *Sensor Security Guard* (SSG), a sensor API hooking and sensitive information filtering framework for the Android platform. For most malicious apps are the culprits for abusing the sensors, so we put SSG right below the application layer in the Android architecture to monitor all potential malicious behavior from above. SSG provides security modes and black/white list mechanism to grant flexible access permissions to the apps. For highly suspicious apps, the access to the sensors are totally prohibited. For normal apps, we use the SSG filters to roughen the sampling from the sensors and return it to the apps. This mechanism ensures the apps cannot calculate accurate results for deriving any privacy from the sensor data.

The main contributions of this work are as follows:

1) *Sensor Security Guard.* We propose and implement a sensor API hooking and sensitive information filtering framework SSG for the Android apps. It's easy to deploy into the system and barely cause performance loss. To the best of our knowledge, SSG is the very first sensor protection system for the Android platform.

2) *Sensor-based attack classification.* The research on sensor-based attacks is so far discrete. Other than focusing on the sensors' functions, we propose a way to classify the attacks according to the resource/privacy the attackers are obtaining from the sensors. Besides proposing the attack categories, we also design five submodules based on the taxonomy for the SSG to handle the corresponding threats from the malicious apps.

## 2 Attack Classification

Each attack would involve more than one sensor. With more sensors, the attack is able to collect more types of information to encompass a precise result. But, no matter how the exploited sensors vary in each attacking scenario, an attack is always focusing on one kind of privacy/sensitive information. Hereby, we classify the attacks according to the types of attack targets they aim at.

- *Location tracing.* The adversary makes use of the sensor data to locate the device without the aid of GPS and network. There are several studies on using motion sensor data to detect user locations [9, 13, 10]. According to [9], accelerometers can be used to locate a device owner within a 200 meter radius of the true location.
- *Sound eavesdropping.* Without access to the microphone on the device, the adversary is still able to collect sound sampling from the gyroscope sensor. From previous works [12, 6], motion sensors will leak sound information. Michalevsky et al. [12] show that the gyroscopes data is sufficient to identify speaker information and even parse speech using signal processing and machine learning.
- *Keystroke monitoring.* The adversary is able to infer user's inputs with the help of sensors. For example, photometer is a light-sensitive sensor. One of the attacks is to infer the keystrokes by measuring the change of the light when the shadow of the figure projecting on the device/photometer. When we make an input with the soft keyboard of the mobile, a great deal of information is passed to the sensors, not only the motion sensors, but also photometers, etc. Similar attacks are introduced in the research like [5, 3, 14]. Besides photometers, the motion sensors are also adopted in such attacks with a high success rate.
- *Device-fingerprinting distinguishing.* With the sensor data, the adversary can get a unique device fingerprint. Some works [7, 4, 8] show that the sensors can also be used to uniquely identify a phone by measuring anomalies in the signals which are the results from manufacturing imperfections. Das et al. [7] not only proposed the method to get a device fingerprint with sensors, but also give the techniques to mitigate such device fingerprinting.
- *User-Identity pinpointing.* The adversary may profile the users using sensor data, even identify the users. According to [11, 15], with the motion sensor's data, we can profile the mobile user and reveal the identity.

## 3 Design of SSG

To avoid the complex mixed sensor attacks in the above scenarios, the naive solution is to block all the sensor data at all time. In this case, a sensor API hooking technique is enough for it intercepts all the information from the sensors. The Android system provides a sensor framework for the developers to access sensor resource. By inserting a hooking module on top of the sensor framework and cut off the sensor data for the apps up-above, API hooking is capable of blocking the potential sensor-based attacks. But, this mechanism is quite rough.

First challenge of designing such type of framework is to filter the suspicious applications accessing the sensor data. For example, a benign sportive app only starts the get-sensor-data process when it's needed, and finishes it immediately when the collecting is over, while a malicious app would keep collecting sensor data as long as it needs. The app filter should be capable to cover the known and unknown attacks of various types. Fortunately, with the attack classification we provided in Section 2, the attacks usually fall into one of the categories.

Another challenge is providing forged data to the apps who overclaim the sensor sampling. It's tightly integrated with the previous challenge. When the security framework identifies the property of the app, it determines a proper accuracy for the purpose of the app.

SSG provides a refined security solution to meet the challenges above. It is capable of processing the corresponding sensor data by the types of the attacks and the credibility of the apps. The overall architecture is shown in Figure 1. It is composed of two modules: 1) a front-end apk file called *SSG Manager*, and 2) a back-end data interception module *Hook Module*.

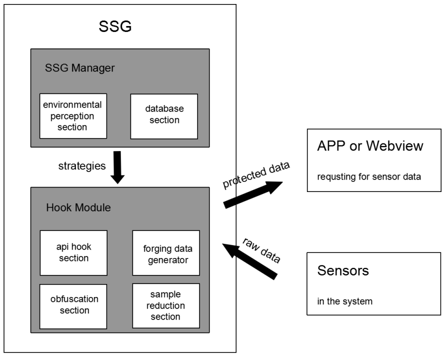


**Fig. 1.** Architecture of SSG

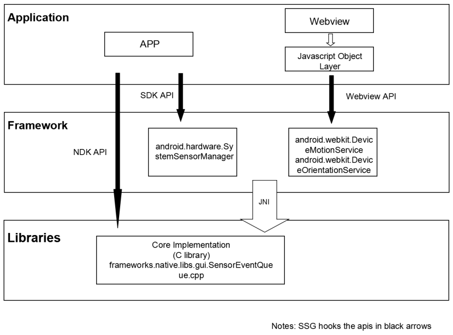

Application
APP
Webview
Javascript Object Layer
SDK API
Webview API
Framework
NDK API
android.hardware.SystemSensorManager
android.webkit.DeviceMotionService android.webkit.DeviceOrientationService
JNI
Libraries
Core Implementation (C library) frameworks.native.libs.gui.SensorEventQueue.cpp
Notes: SSG hooks the apis in black arrows

**Fig. 2.** Sensor Data Sources in Android

### 3.1 SSG Manager

The front-end *SSG Manager* decides how to process the sensor data according to the apps, a.k.a. the data processing strategy provider. *SSG Manager* has two tasks: 1) The app user decides how to apply the strategy. We provide three security modes for the apps, and pass the right to the users who can make the decisions through a graphic user interface. 2) It chooses the protection strategies accordingly by perceiving the status of the smartphone, see Table 1. For instance, when the user inputs password from the keyboard, any access to the pressure and sound related sensor data is forbidden.

Based on the attack classification, we have collected most threat scenarios and provided corresponding countermeasures. It is implemented in the submodule *Environmental Perception Section.*

SSG manager also maintains the black/white list of the application, in the submodule *Database Section.* The database is built on the choice of the user– put an app into black or white list. If the user could not decide by herself at the moment, SSG manager will decide the security strategy based on the environmental information. This database is growing with the time. Maintaining such a database on the cloud-end is efficient to respond to emerging attacks.

### 3.2 Hook Module

The *Hook Module* works as a sentry. By hooking the sensor APIs, implemented by submodule *API Hook Section*, it passes on the data collection requirements from the apps and WebView, then gathers all the data from the sensors underneath. Besides, *Hook Module* provides the optional functionality of roughening the sampling data from the sensors in three submodules *Forging Data Generator*, *Obfuscation Section* and *Sample Reduction Section.* By doing so, we interrupt the accuracy of the data that is required to recover the privacy/sensitive information.

### 3.3 SSG Work flow

A typical processes of SSG is as follows: when the system launches, SSG traverses all the apps that are using the sensors. The user can manually configure the security mode for each app. There are three modes: *Secure Mode*, *Free Mode* and *Normal Mode.*

When the user requires a highly secure operation (such as online payment, private phone call), he can choose the *Secure Mode* for the specific app. Then, all types of sensors' data are forged by *Hook Module* and provided to all the apps that are using sensor data at this moment. For example, during an online payment transaction, *Secure Mode* disables the access to the sensor system-wide, so that the malicious apps, if any, running at the background cannot gather any sensor data.

If the user chooses *Free Mode* when he is using a sensor-related app in a less privacy-sensitive environment. When an app or a WebView asks for any type of sensors data, SSG will provide the raw sampling data without any process. However, in this mode, the user is at high information leak risk on his own choice.

At last, all the other apps will be under *Normal Mode* by default. SSG provides an even fine-grained black/white list for each app. If it's the first time that an app asks for sensor data, SSG will ask the user to put this app into black list, white list or neither. SSG provides raw data for the apps in the white list. On the contrary, the apps in the black list are provided with the forging data from *Hook Module*. If the apps are on neither lists, SSG will provide context-sensitive protection strategies according to the state given in Table 1. By perceiving the mobile context, SSG applies different countermeasures against various threat scenarios. Detailed description can be found in Section 5.1.

## 4 Implementation

At designing stage, SSG consists of two major modules: *Hook Module* and *SSG Manager*. In *Hook Module*, there are four submodules: *API Hook Section*, *Forging Data Generator*, *Obfuscation Section*, and *sample Reduction Section*. *SSG Manager* is composed of *Environmental Perception Section* and *Database Section*.

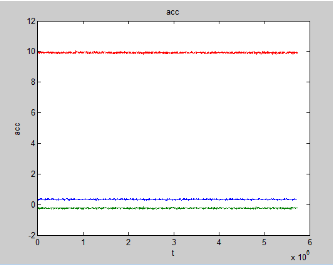
We implement SSG in Android 4.2.2 on the device of Nexus 4. The *Hook Module* relies on Cydia Substrate [1], and should be pre-put into the system image or installed with root privilege. For the *SSG Manager*, we develop an app which should be installed as a common one. Next, we will come to the specific introduction of each section's implementation.

### 4.1 Hook Module

**API Hook Section** Figure 2 indicates the residence of SSG in the Android architecture.The Android system provides two ways to access the sensors data on the device: Application and WebView.

**Application**: Most of local apps will access sensors by using the Android sensor framework, or from NDK using C or C++. An app can get an instance of *android.hardware.Sensor* by the method *getDefaultSensor* of the package *android.hardware.SensorManager*. Then call the method *SensorManager.registerListener*, which will attach *android.hardware.SensorEventListener* to the *Sensor*. At last, we achieve *SensorEventListener*'s callback function *onSensorChanged* to obtain the sensor data. In this way, we can access all kinds of sensors available on the device. *hardware/libhardware/include/hardware/sensor.h* shows the API provided by the NDK. */frameworks/native/libs/gui/ SensorEventQueue.cpp*, *Sensor.cpp* and *SensorManager.cpp* give details of the API methods. We can get a Sensor instance by *SensorManager.getDefaultSensor*. Then we attach *SensorEventQueue* to *Sensor* and obtain the data from method *SensorEventQueue.read*.

**WebView**: In a WebView, we can access the sensor with Javascript handler: *window.ondevicemotion* and the event object: *event.accelerationGravity*, *event.accelerationIncludingGravity* and *event.rotationRate*. In this way, we can only access limited kinds of sensors.



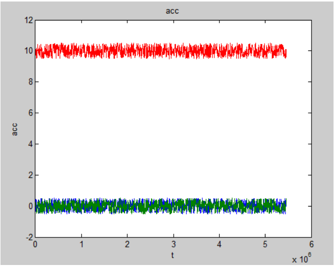
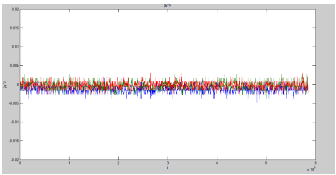


**Fig. 3.** raw data of accelerometer while the device is static

**Fig. 4.** forging data of accelerometer



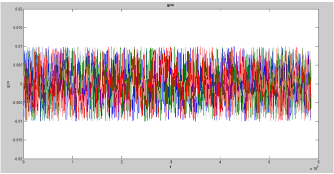


**Fig. 5.** raw data of gyroscope while the device is static

**Fig. 6.** forging data of gyroscope

**Forging Data Generator** SSG replaces the raw sensor data with forging data under different strategy choices. The forging data would better not contain any information about the specific device, the user or the ambient information at the moment. For various sensors, different rules are formulated. SSG forges the data which makes the device looks static. For example, the forging accelerometer data: axis $x$, axis $y$, will be limited in the range [0.5, -0.5], and axis $z$ will be limited in range [9.5, 10.5]. The forging gyroscope data: $x$,$y$, and $z$ will be limited in range [0.01, -0.01]. We show the forging data examples in Figure 3,Figure 4,Figure 5 and Figure 6.

**Obfuscation** In order to increase the difficulty of obtaining the fingerprint information and reduce the accuracy rate, SSG chooses Anupam Das Basic Obfuscation [7] to add noise and hide the anomalies. The details of the obfuscation is same as the paper [7]'s Basic Obfuscation. We also compute $a_O = a_M * g_O + o_O$, where $g_O$ and $o_O$ are the obfuscation gain and offset, respectively. We choose a range of [-0.5, 0.5] for the accelerometer offset, [-0.1,0.1]for the gyroscope offset, and [0.95,1.05] for the gain.

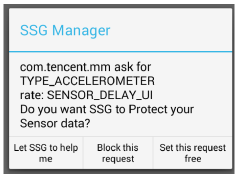
**Sampling Reduction** In order to prevent high frequency signals leak from the gyroscope and accelerometer, SSG limits the data acquisition frequency of the two sensors. Consider that a high frequency may provide the attacker accurate information, SSG decreases the highest sampling rate by adding 0.02s delay at least. As we know, the highest sampling frequency of gyroscope is 200Hz in Android, so the least delay will be 0.005s. After decreasing the sampling rate, we know that the least delay in SSG is 0.025s, so the highest frequency can only reach 40Hz.

### 4.2 SSG manager

**Environmental Perception** Environmental perception helps to decide the security strategy of the apps that are under *Normal Mode* in *SSG Manager*, most security strategies depend on the actual smartphone environment as listed in Table 1. According to the actual environment, SSG takes different measures to protect. The environment perception is important and its responsibility is to perceive the mobile environment, and pass the strategy related with the environment to the *Hook Module*.

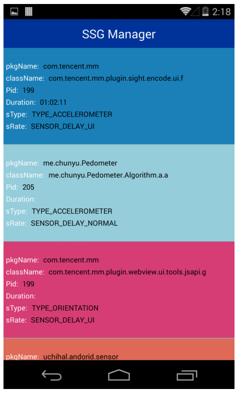
| Perceived behavior | Method | Permission |
|---|---|---|
| velocity of the device | API:<br>android.location.LocationManager.<br>requestLocationUpdates | ACCESS_COARSE_LOCATION<br>ACCESS_FINE_LOCATION |
| Call state of the device | API:<br>android.telephony.TelephonyManager<br>.getCallState()<br>Broadcast Receiver<br>Intent.ACTION_NEW_OUTGOING_CALL | READ_PHONE_STATE |
| Soft keyboard state of the device | Hook:<br>android.inputmethodservice.InputMethodService<br>.showWindow<br>android.inputmethodservice<br>.InputMethodService.doHideWindow | Root |
| Screen state of the device | API:<br>android.os.PowerManager.isScreenOn();<br>android.app.KeyguardManager<br>.inKeyguardRestrictedInputMode() | None |

**Table 1.** Environmental Factors Monitored by SSG


SSG Manager

com.tencent.mm ask for
TYPE_ACCELEROMETER
rate: SENSOR_DELAY_UI
Do you want SSG to Protect your
Sensor data?

**Fig. 7.** SSG Manager's toast



SSG Manager

pkgName: com.tencent.mm
className: com.tencent.mm.plugin.sight.encode.ui.f
Pid: 199
Duration: 01:02:11
sType: TYPE_ACCELEROMETER
sRate: SENSOR_DELAY_UI

pkgName: me.chunyu.Pedometer
className: me.chunyu.Pedometer.Algorithm.a.a
Pid: 205
Duration:
sType: TYPE_ACCELEROMETER
sRate: SENSOR_DELAY_NORMAL

pkgName: com.tencent.mm
className: com.tencent.mm.plugin.webview.ui.tools.jsapi.g
Pid: 199
Duration:
sType: TYPE_ORIENTATION
sRate: SENSOR_DELAY_UI

pkgName: uchihal.andorid.sensor

**Fig. 8.** SSG Manager's information interface

**Database Section: Black/White List Managing** This section is responsible for the storage of user configuration rules and the current use states of sensors. All data is stored in the *SSG Manager*'s database (SQLite3). When an app requests sensor data for the first time, the SSG will ask the user whether needs to protect the sensor data. At this point, the user has three options, as shown in Fig7:

1. Set this request free — Add the app to the white list, provide real data later.
2. Block this request — Add the app to the black list, provide forging data later.
3. Let SSG to help me — Add security strategy to the app, provide different strategies according to the different environment.

In addition to the black and white list, this section will also store the use states of the sensors. Record form is as follows:

*pkg name:class name:pid:start time:sensor type:sensor rate*

E.g. : *com.tencent.mm:com.tencent.mm.plugin.sight.encode.ui.f:199:1439374073896:1:2*

These status records will be displayed to the user through the *SSG Manager*'s graphic user interface as shown in Fig8.

# 5 Evaluation

## 5.1 Security

- **Location Protection**A benign app only starts the get-sensor-data process when it's needed, and will finish it immediately when the collecting is over. According to our observation, the limited time is suggested to be no more than 20 min [9]. In our experiments, when the device velocity is over 20km/h, SSG starts to provide forging velocity data periodically.
- **Sound Noising** When the smarphone is at a call state, SSG provides forging data to all non-system apps. The forging data contains no useful information obviously, and it will not affect any app's function. If the phone is not in a call state, SSG will limit the sensors with a sampling frequency under 160Hz in default. According to the Nyquist sampling theorem, a sampling frequency f enables us to reconstruct signals at frequencies of up to f /2. So we can say that we can only completely reconstruct signals under 80Hz from the gyroscope data which will not contain any information about the human speech.
- **Keystroke Shield** We find that when the soft keyboard is on or the phone is in a waiting-for-unlocking state, no sensor is required by any app in general. SSG will provide forging data in such cases and will not affect any app's function.
- **Device-Fingerprinting Blurring** According to [7, 4, 8], attacker can get a unique fingerprinting with the sensor data because that sensor may have anomalies in the signals and different devices may have different features of anomalies. The paper gives two ways to mitigate such device fingerprinting: calibrating and obfuscation. Das et al. have verified its effectiveness in his paper, so we use it directly, not to prove.
- **User-Identity Protection** Identity attack has the same features as the location attack. So the strategy that limit the time of sensor data collection is equally effective here. In particular, most apps shouldn't use the sensors in the background. Therefore, if the app is pushed to the back ground, SSG will minus the limited time(e.g. default 10 mins). This will make the data collecting work much more difficult.

### 5.2 Usability

| App | Sensors used |
|---|---|
| Wechat | *accelerometer,orientation,proximity* |
| QQ | *accelerometer,gyroscope,light,pressure,proximity,gravity* |
| BaiduMap | *accelerometer,magnetic-field,orientation,gyroscope, pressure,gravity,linear-acceleration,rotation-vector* |
| AutonaviMap | *accelerometer,magnetic-field,orientation,gyroscope, pressure* |
| TencentMap | *accelerometer,gyroscope,light,pressure, gravity* |
| SogouMap | *accelerometer,magnetic-field,orientation,gyroscope, rotation-vector* |
| TencentPao | *accelerometer,step-counter* |
| SSGame | *accelerometer,gyroscope,light,gravity* |
| Kuwo | *accelerometer,magnetic-field,light* |
| zhanqiAndroid | *accelerometer,gyroscope,proximity,linear-acceleration* |
| Dongdong | *orientation,proximity,step-counter* |
| Runtastic | *accelerometer,magnetic-field,orientation,proximity* |
| XiaomiHealth | *accelerometer,orientation,step-counter* |
| Bamboo | *accelerometer,magnetic-field,orientation,light, pressure,proximity,step-counter* |
| Taobao | *accelerometer,gyroscope,gravity* |

**Table 2.** Selected apps in 30 test experiment objects

First, we validate that SSG can effectively provide the sensor data to the common apps and verify that it will not cause the apps to crash. We select 30 apps with high download rates in 6 categories from the app markets. Check app's use state of sensors by static analysis. Then manually install, run and verify the apps on Nexus 4. The results in table 2 show that every app can run normally without any crash.

### 5.3 Effectiveness

To evaluate whether the sensor data provided by SSG can hide the user privacy, we consider 3 scenarios: inputting , calling , moving.

For input, we manually type some characters using system soft keyboard. Meanwhile, we collect and record the raw sensor data provided by system and the data provided by SSG respectively. Then we transform the data into images by Matlab as shown in Figure 9 and Figure 12. For phone call, we manually use the phone to make a call and record the raw sensor data and forging data respectively as before. The result is shown as in Figure 10 and Figure 13. For device moving, we drive at speeds of over 20 kilometers an hour with the phone in the car and test two cases: In one hand, we test the case when the duration of get-sensor-data process is over the limited time as shown in Figure 11. In another, we turn on the GPS of the phone and test the case with the duration within the limited range as shown in Figure 14. SSG has succeeded in smoothening the spiky curves. Therefore, the attackers are not able to extract any useful information.

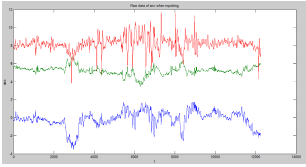


**Fig. 9.** raw data of accelerometer on input

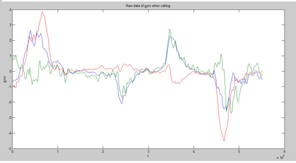


**Fig. 10.** raw data of gyroscope on call

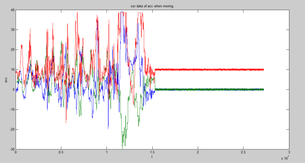


**Fig. 11.** forging data of accelerometer when the duration of get-sensor-data process is over the limited time

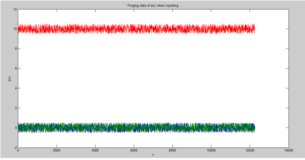


**Fig. 12.** forging data of accelerometer on input

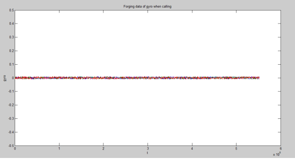


**Fig. 13.** forging data of gyroscope on call

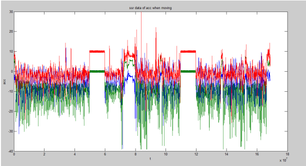


**Fig. 14.** forging data of accelerometer while driving at speeds of over 20 kilometers an hour

### 5.4 Cost

There are two reasons to explain that SSG will not have too much energy cost: For one thing, there is no complex calculations in SSG and the energy cost of SSG is almost as much as a common app's. For another, SSG works only when some apps are requesting for sensor data. And some functions of SSG (e.g. Generating forging data) need certain context(e.g. The keyboard is on, the phone is in a calling state and so on). However, the time when sensors are working has a really small proportion of the total time, not to mention the certain context.

To evaluate the energy overhead, we develop a test app to request for sensor(accelerometer) data continuously. Then we install the test app into the Nexus 4 device with the battery in a full state. There is nothing installed and running in the device but SSG and the test app. We consider three scenarios: no sensor access, one sustained load of sensor (accelerometer) access, one sustained load of sensor (accelerometer) access with SSG providing forging data. We compare the time when the battery is running down. During the experiment, We tried to shut down all the power consumption sections of the device,such as screen, WiFi, GPS,MONET and so on. The results show that there is nearly no difference whether the SSG is on. The experimental data can't point account for the actual energy cost by the SSG, however, to a certain extent, it can be explained that SSG will not have a high energy cost.

## 6 Conclusion

In this paper, we analysised the security problems in the sensors of Android and firstly proposed the concept of sensor protection.Then we proposed SSG(sensor security guarder)–a hook-based sensor protection framework for Android smartphones which can practically, effectively and efficiently protect the sensors from kinds of attacks including location attack, speech attack, keystroke attack, device-fingerprinting attack and user-identity attack. We fully implemented SSG in Android 4.2.2 on Nexus 4. It's not difficult to deploy and compatible with all the 30 test-apps which all have sensor-related functions and high download rates. We have measured the protection effects of SSG in three scenarios: inputting, calling, moving. The results showed that SSG has an obviously protective effect and incurs acceptable energy.

## References

1. Cydia substrate. `http://www.cydiasubstrate.com/`.
2. Nike+ running applications. `http://www.nike.com/us/en\_us/c/running/nikeplus/gps-app`.
3. A. Al-Haiqi, M. Ismail, and R. Nordin. On the best sensor for keystrokes inference attack on android. *Procedia Technology*, 2013.
4. H. Bojinov, Y. Michalevsky, G. Nakibly, and D. Boneh. Mobile device identification via sensor fingerprinting. *arXiv preprint arXiv:1408.1416*, 2014.
5. L. Cai and H. Chen. Touchlogger: Inferring keystrokes on touch screen from smartphone motion. In *In Proc.Of HotSec, 6th*, 2011.
6. D. Currie. Shedding some light on voice authentication, 2009.
7. A. Das, N. Borisov, and M. Caesar. Exploring ways to mitigate sensor-based smartphone fingerprinting. *arXiv preprint arXiv:1503.01874*, 2015.
8. S. Dey, N. Roy, W. Xu, R. R. Choudhury, and S. Nelakuditi. Accelprint: Imperfections of accelerometers make smartphones trackable. In *In Proc.Of the Network and Distributed System Security Symposium (NDSS), 21st*, 2014.
9. J. Han, E. Owusu, L. T. Nguyen, A. Perrig, and J. Zhang. Accomplice: Location inference using accelerometers on smartphones. In *In Proc.Of Communication Systems and Networks (COMSNETS), 4th*, 2012.
10. S.-W. Lee and K. Mase. Activity and location recognition using wearable sensors. *IEEE pervasive computing*, 2002.
11. J. Mäntyjärvi, M. Lindholm, E. Vildjiounaite, S.-M. MäkelIEEEä, and H. Ailisto. Identifying users of portable devices from gait pattern with accelerometers. In *In Proc.Of Acoustics, Speech, and Signal Processing(ICASSP'05), 30th*, 2005.
12. Y. Michalevsky, D. Boneh, and G. Nakibly. Gyrophone: Recognizing speech from gyroscope signals. In *In Proc.Of USENIX Security Symposium, USENIX Association, 23rd*, 2014.
13. P. Mohan, V. N. Padmanabhan, and R. Ramjee. Nericell: rich monitoring of road and traffic conditions using mobile smartphones. In *In Proc.Of ACM conference on Embedded network sensor systems, 6th*, 2008.
14. R. Spreitzer. Pin skimming: Exploiting the ambient-light sensor in mobile devices. In *In Proc.Of ACM Workshop on Security and Privacy in Smartphones & Mobile Devices, 4th*, 2014.
15. H. Wang, D. Lymberopoulos, and J. Liu. Sensor-based user authentication. In *In Proc.Of Wireless Sensor Networks, 7th.* 2015.