2019 IEEE Symposium on Security and Privacy

SENSORID: Sensor Calibration Fingerprinting for Smartphones

Jiexin Zhang, Alastair R. Beresford
University of Cambridge
{jz448, arb33}@cl.cam.ac.uk

Ian Sheret
Polymath Insight Limited
ian.sheret@polymathinsight.co.uk

***Abstract*—Sensors are an essential component of many computer systems today. Mobile devices are a good example, containing a vast array of sensors from accelerometers and GPS units, to cameras and microphones. Data from these sensors are accessible to application programmers who can use this data to build context-aware applications. Good sensor accuracy is often crucial, and therefore manufacturers often use per-device factory calibration to compensate for systematic errors introduced during manufacture. In this paper we explore a new type of fingerprinting attack on sensor data: *calibration fingerprinting*. A calibration fingerprinting attack infers the per-device factory calibration data from a device by careful analysis of the sensor output alone. Such an attack does not require direct access to any calibration parameters since these are often embedded inside the firmware of the device and are not directly accessible by application developers. We demonstrate the potential of this new class of attack by performing calibration fingerprinting attacks on the inertial measurement unit sensors found in iOS and Android devices. These sensors are good candidates because access to these sensors does not require any special permissions, and the data can be accessed via both a native app installed on a device and also by JavaScript when visiting a website on an iOS and Android device. We find we are able to perform a very effective calibration fingerprinting attack: our approach requires fewer than 100 samples of sensor data and takes less than one second to collect and process into a device fingerprint that does not change over time or after factory reset. We demonstrate that our approach is very likely to produce globally unique fingerprints for iOS devices, with an estimated 67 bits of entropy in the fingerprint for iPhone 6S devices. In addition, we find that the accelerometer of Google Pixel 2 and Pixel 3 devices can also be fingerprinted by our approach.**

***Index Terms*—Fingerprint; Calibration; Motion Sensors; iOS**

I. INTRODUCTION

Mobile devices, especially smartphones and tablets, have gained considerable popularity due to their portability, user experience, and rich functionality. Recent statistics show that around 69% of the total digital media time spent by Americans was taken up by mobile devices and 87% of the total mobile time was spent in apps in December 2016 [1]. The faltering engagement with laptops and desktops has driven advertisers to enthusiastically seek effective approaches to track the activities of users on mobile platforms.

Device fingerprinting aims to generate a distinctive signature, or *fingerprint*, that uniquely identifies a specific computing device. Once a reliable device fingerprint is available, advertisers can use it to track users online and offline, study their behavior, deliver tailored content, etc. It can also help advertisers target potential customers and grow their business.

To protect user privacy, both Android and iOS have applied a variety of measures to prevent device fingerprinting. In Android, the countermeasures are usually enforced by requiring developers to ask for user permission to use privacy-sensitive APIs. For instance, the `READ_PHONE_STATE` permission enables developers to access the IMEI (International Mobile Equipment Identity) number and the phone number. The `ACCESS_WIFI_STATE` and `BLUETOOTH` permissions allow developers to access the MAC address of the WLAN and Bluetooth modules, respectively. These identifiers make good device fingerprints since they are globally unique and are difficult to change, and a recent study has shown that the `ACCESS_WIFI_STATE` permission, an install-time permission, has been used in 41% of 2700 most popular apps in the Google Play Store [2].

Currently, there is no reliable way to get a unique identifier for iOS devices. To address privacy concerns, Apple removed developer access to the UDID (Unique Device Identifier) and MAC addresses of hardware modules in iOS 7. From iOS 11, Apple introduced Intelligent Tracking Prevention (ITP) to mobile Safari, the default web browser on iOS that holds around 15% of the global browser market, to restrict cookie tracking. With iOS 12, Apple also prevents advertisers from collecting unique characteristics of iOS devices through mobile Safari.[1] While it is still possible to track iOS users by the advertising identifier built into the operating system, this method comes with several drawbacks. First, iOS allows users to reset the advertising identifier at any time. In addition, enabling the "limit ad tracking" option in iOS prevents access to this identifier. Moreover, apps that request this identifier but do not serve any in-app advertisements will be rejected by App Store. Last but not least, the advertising identifier is not accessible from mobile browsers. Thus, it cannot be used to track users across apps and websites.

In this paper we show that our calibration fingerprinting attack on iOS and Android (Google Pixel 2 and Pixel 3) devices provides a device fingerprint which the user cannot change and which works across both apps and websites.

Modern mobile devices are shipped with a variety of embedded sensors, such as an accelerometer, gyroscope, and

[1] https://www.apple.com/ios/ios-12/features

DOI 10.1109/SP.2019.00072

magnetometer. Mobile apps rely on these sensors to provide rich functionality, such as workout tracking, improved user interface interaction, and better gaming performance. Natural variation during the manufacture of embedded sensors means that the output of each sensor is unique and therefore they may be exploited to create a device fingerprint.

Previous studies applied machine learning techniques directly to sensor data in an attempt to create device fingerprints for smartphones [3], [4]. Unfortunately, this approach has several drawbacks. First, these models are susceptible to environmental conditions, such as temperature and noise, so accuracy decreases over time. Second, they either require the smartphone to be placed in a stationary position (e.g., on a desk) or have a relatively low accuracy when devices are moved. When devices were held in a hand, state-of-the-art work by Das *et al.* achieved an accuracy of around 60% in an open-world setting with the iPhone 6, which is equivalent to 13 bits of entropy (§VII-D) [4]. In comparison, we achieve 67 bits of entropy for the iPhone 6S.

In our work, we take a different approach. Instead of feeding sensor outputs into machine learning algorithms, we infer the per-device factory calibration data from the output of sensors such as the gyroscope, accelerometer, and magnetometer. This calibration data can then be used to construct a unique device fingerprint. We call this a *calibration fingerprinting attack*.

This paper describes this new type of attack and demonstrates its effectiveness on gyroscope and magnetometer data available in iOS, and on accelerometer data in Google Pixel 2/3. We chose these sensors because they do not require any special permissions to access, and the accelerometer and gyroscope data can be accessed via both a native app installed on a device and also by JavaScript when the user visits a website. While we have focused on motion sensors, we anticipate that a calibration-based fingerprint can also be generated for other sensors across many different devices, including the camera, touchscreen, and battery.

We make the following contributions in this paper:

1) We introduce a new method of fingerprinting a device: the calibration fingerprinting attack.
2) We describe how factory calibration data can be extracted from both the magnetometer and gyroscope found on all recent iOS devices.
3) We demonstrate that the magnetometer and gyroscope calibration data together form a reliable fingerprint, which we call the SENSORID; the SENSORID does not change after factory reset or operating system update.
4) We collect motion sensor data from 870 iOS devices and show that our approach can generate a globally unique identifier; we show that the SENSORID of the iPhone 6S has about 67 bits of entropy.
5) We implement our approach as an iOS app and find the approach is lightweight and efficient: data collection and processing typically takes less than one second in total.
6) We find that the accelerometer of Google Pixel 2 and Pixel 3 can be fingerprinted using the proposed method.
7) We propose several mitigation techniques.

II. CALIBRATION BACKGROUND

Motion sensors used in modern smartphones, including the accelerometer, gyroscope, and magnetometer, are based on MEMS (Micro-Electro-Mechanical Systems) technology and use microfabrication to emulate the mechanical parts. The accelerometer and gyroscope measure the proper acceleration and rotation speed of a device in each of the axes, respectively. Most smartphones and smartwatches are equipped with one triaxial accelerometer and one triaxial gyroscope. In particular, gyroscope has appeared in every iOS device since the iPhone 3GS and iPad 1. The magnetometer measures the Earth's magnetic field relative to the device. It is included in all iPhone and iPad models, excluding the iPhone 2G/3G.

Although MEMS technology has greatly reduced the size and cost of motion sensors, MEMS sensors are usually less accurate than their optical counterparts due to various types of error. In general, these errors can be categorized as *deterministic* and *random*: random errors are usually caused by electronic noise interfering with the output of sensors, which change over time and have to be modeled stochastically; deterministic errors are produced by manufacturing imperfections and can be classified into three categories: *bias*, *scaling factor*, and *nonorthogonality misalignment errors* [5], [6].

Calibration aims to identify and remove the deterministic errors from the sensor. Many commercial sensors are factory calibrated and their calibration parameters are stored in firmware or non-volatile memory, providing accurate measurements off the shelf [7]. In the context of mobile devices, the main benefit of per-device calibration is that it allows more accurate attitude estimation [8]. By contrast, sensors embedded in low-cost smartphones are usually poorly calibrated due to the high cost and complexity of factory calibration [9]. For an individual manufacturer, the choice of sensor calibration is, therefore, an engineering trade-off.

MEMS sensors usually convert and store the analog measurement in a digital register through an Analog-to-Digital Converter (ADC) module. For a triaxial motion sensor, let $\mathbf{A} = [A_x, A_y, A_z]^T$ be the sensor ADC output. Considering all three kinds of deterministic errors, the output of the motion sensor can be represented by the following equation [10]:

$$\begin{bmatrix} O_x \\ O_y \\ O_z \end{bmatrix} = \begin{bmatrix} S_x & 0 & 0 \\ 0 & S_y & 0 \\ 0 & 0 & S_z \end{bmatrix} \begin{bmatrix} N_{xx} & N_{xy} & N_{xz} \\ N_{yx} & N_{yy} & N_{yz} \\ N_{zx} & N_{zy} & N_{zz} \end{bmatrix} \begin{bmatrix} A_x \\ A_y \\ A_z \end{bmatrix} + \begin{bmatrix} B_x \\ B_y \\ B_z \end{bmatrix} \quad (1)$$

Here, $S_i \in \mathbf{S}$ is the scale factor; $N_{ij} \in \mathbf{N}$ represents the nonorthogonality between axis i and j; and $B_i \in \mathbf{B}$ is the bias. A sensor's *sensitivity*, or *gain*, is defined as the ratio between the output signal and measured property. A sensor's nominal gain is the intended operating sensitivity of the sensor. It is a single value that is usually documented in the sensor datasheet. We use F to denote a sensor's nominal gain in this paper. If a sensor is ideal, its scale matrix $\mathbf{S}$ and nonorthogonality matrix $\mathbf{N}$ should be $F \cdot \mathbf{I}$ and $\mathbf{I}$, respectively, where $\mathbf{I}$ is an identity matrix. However, due to the existence of errors, the

scale factors can be as large as 2% of the nominal gain [11]. The above equation can be further simplified as:

$$\mathbf{O} = \mathbf{GA} + \mathbf{B} \tag{2}$$

where $\mathbf{G} = \mathbf{SN}$ is referred to as the gain matrix.

A myriad of calibration techniques have been proposed to calculate the gain matrix and bias vector during manufacture. Overall, these methods can be divided into four groups: *high-precision equipment*, *multi-position*, *Kalman filter*, and *vision based* [5]. Manufacturers can choose to only calibrate the bias vector to lower the cost. Once factory calibration is finished, the calibration parameters of the sensor will be stored in non-volatile memory inside the device and should not change over time [12], [13]. Details of the calibration process used by manufacturers are not made public.

III. Attack Method

The goal of the adversary in this paper is to obtain a reliable fingerprint from the built-in motion sensors of a smartphone. Our threat model is as follows. We assume an adversary is able to record motion sensor samples from a smartphone. The attacker can do this if the user installs an app, or visits a website (currently accelerometer and gyroscope only), under the control of the attacker. Furthermore, we assume that the software embedded in the app or web page is able to communicate with a remote server under the control of the attacker; this is typically the case for both apps and web pages. We first look at the gyroscope on iOS; the calibration fingerprinting for other sensors and devices is discussed later.

A. Factory Calibration in Mobile Devices

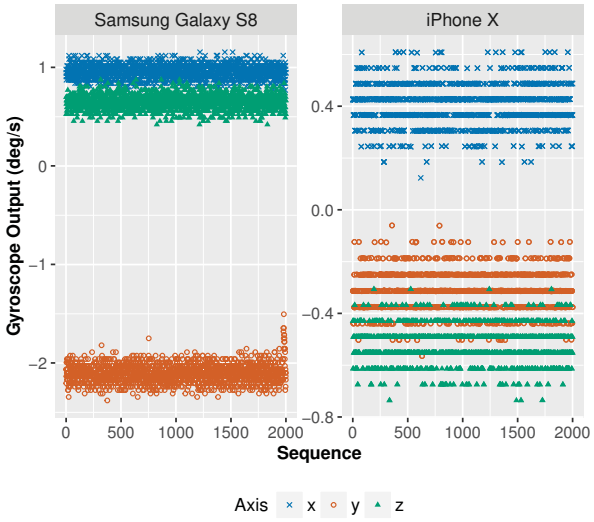
Both Android and iOS provide APIs to access the *raw* and *fused* motion sensor data. Web developers can also access the *fused* accelerometer and gyroscope data via JavaScript. According to the Android documentation, the *raw* or *uncalibrated* data is the sensor output after factory calibration and temperature compensation, while the *fused* or *calibrated* sensor data API applies bias compensation and noise correction on the raw measurements.[2] On iOS, it is less clear from the documentation whether the raw data is factory calibrated. To investigate this, we collect the raw gyroscope data from both an iPhone X and a Samsung Galaxy S8. Both phones are placed on a flat desk and stay still during data collection.

Fig. 1 (a) presents the raw gyroscope measurements collected by the two devices. From the figure, we can clearly observe the quantization. This is because the outputs of the gyroscope ADC are integers. Taking the difference between two sensor readings directly reveals the gain of the sensor. According to Equation 2, the difference between two measurements, $\Delta\mathbf{O}$, can be calculated as:

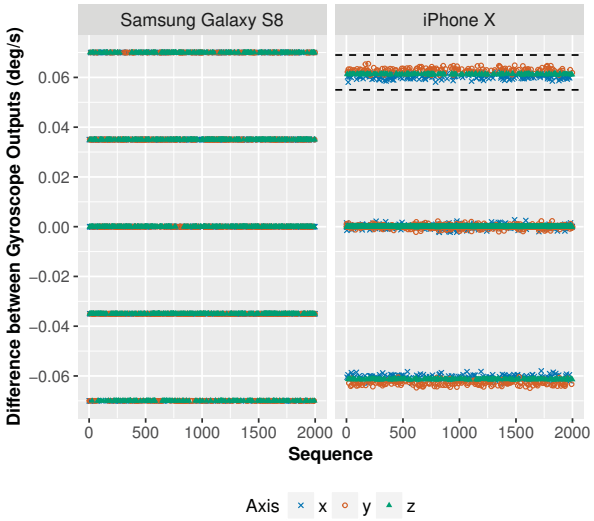
$$\Delta\mathbf{O} = \mathbf{G}\Delta\mathbf{A} \tag{3}$$

where $\Delta\mathbf{A}$ is the difference between the corresponding ADC outputs.

[2]https://developer.android.com/guide/topics/sensors/sensors_motion.html



(a) Raw gyroscope measurements



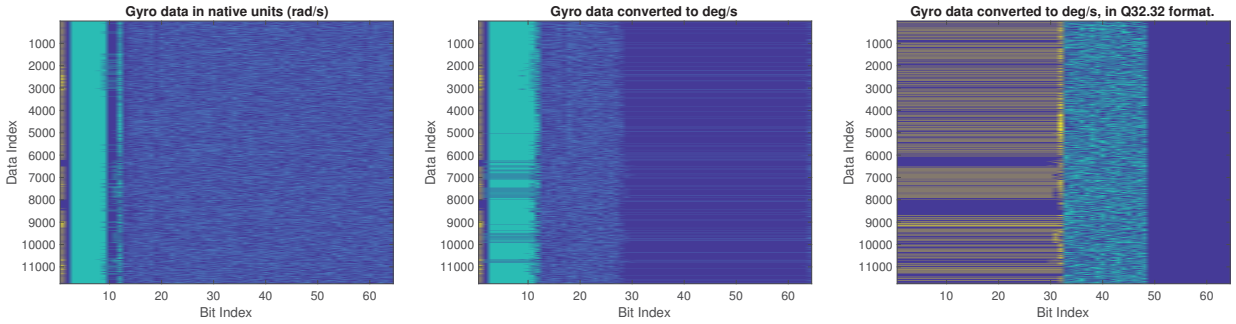
(b) Consecutive differences between gyroscope measurements

Fig. 1: Raw gyroscope data collected from a Samsung Galaxy S8 and an iPhone X

If there is no factory calibration, the gain matrix $\mathbf{G}$ will be equal to the $F_G \cdot \mathbf{I}$, where F_G is the nominal gain of the gyroscope. In this case, we will have:

$$\begin{bmatrix}\Delta O_x \\ \Delta O_y \\ \Delta O_z\end{bmatrix} = F_G \begin{bmatrix}\Delta A_x \\ \Delta A_y \\ \Delta A_z\end{bmatrix} \tag{4}$$

Since ΔA_i is an integer, ΔO_i should be a multiple of the nominal gain. Fig. 1 (b) shows the difference between consecutive measurements for each of the three axes. Notably, the figure only presents a small range of data to show the results more clearly. As seen in Fig. 1 (b), the difference for the



(a) rad/s, IEEE 754 format (b) deg/s, IEEE 754 format (c) deg/s, Q32.32 format

Fig. 2: Different binary representations of the raw gyroscope data

Samsung Galaxy S8 is always a multiple of a constant value (nominal gain). By contrast, the differences for the iPhone X are not a single multiple. Fig. 1 also reveals that each axis of the gyroscope in the iPhone X has a different gain and bias (the `x` axis has a slightly lower gain than the `y` and `z` axes).

Overall, Fig. 1 demonstrates two things: (i) the gain matrix of the gyroscope in the iPhone X is factory calibrated while the one in the Samsung Galaxy S8 is not; and (ii) the iOS API for accessing raw sensor data obtains the factory calibrated data. We further implement the same experiment on other iOS device models and confirm their gyroscope is also factory calibrated. The factory calibration of motion sensors in Android devices is discussed in §V-D.

B. Nominal Gain Estimation

In general, manufacturing imperfections introduce idiosyncrasies across different sensors. If factory calibration is carried out on a per-device basis, then the calibration matrices, which include the gain matrix $\mathbf{G}$ and the bias matrix $\mathbf{B}$, may also be unique. Therefore, if we can recover any of these matrices, then it may be used as a device fingerprint. In this paper we focus on recovering the gain matrix $\mathbf{G}$. We first investigate fingerprinting devices from mobile apps, where raw sensor data is accessible.

To recover the gain matrix $\mathbf{G}$, we need to know the nominal gain of each device. For some earlier iOS devices, such as iPhone 4, the nominal gain of the gyroscope, 70 millidegrees per second (mdps), is specified in the datasheet. Although we could not find the gyroscope specification for recent iOS device models, we can estimate the nominal gain from gyroscope measurements.

Equation 3 shows that the bias $\mathbf{B}$ can be eliminated by taking the difference between two sensor outputs. From Fig. 1 (b), we observe that the actual gain of each axis is in close proximity to each other (or to the nominal gain) and the fluctuation within each cluster is small compared with the large gap between different clusters. This implies that the actual gain matrix is close, but not equal, to the ideal gain matrix ($F_G \cdot \mathbf{I}$). Since the iPhone X was resting on the desk during data collection, the difference between consecutive ADC outputs ΔA_i should be small (e.g., 0, ± 1). To estimate the nominal gain (F_G) of the iPhone X, we find the cluster with only positive values that is closest to 0 but its range does not include 0. In the case of Fig. 1 (b), the qualified clusters is the one inside dashed black lines. Then the average of all the values inside the cluster, which includes data from all three axes, can be used as a reliable estimate of the nominal gain of the iPhone X. Note that this approach requires the device is stationary (e.g., at rest on a desk) during measurement so that we have enough data points with $\Delta A_i = \pm 1$. This is usually not a problem since we only need to estimate the nominal gain once for each device model. Appendix A shows the estimated nominal gain from two iPhone X devices using this approach.

TABLE I: Estimated gyroscope nominal gain for iOS devices

Model	Nominal Gain (mdps)
iPhone 5S*/6/6 Plus/6S/6S Plus/7/7 Plus/8/8 Plus/SE iPhone X/XS/XS Max iPad Pro 9.7/10.5/12 inch	61
iPhone 4/4S/5/5C/5S*, iPad 3/Mini/Mini 4/Mini Retina/Air/Air 2	70

* iPhone 5S devices have two possible nominal gain values.

Table I lists the nominal gain (in mdps) of the gyroscope for all the iOS devices that we have measured. Note that we have observed two possible nominal gain values for iPhone 5S devices. This may be because some iPhone 5S devices use a different IMU model from others. The estimated nominal gain of 61 mdps indicates that the sensor is likely configured to a measurement range of ± 2000 dps and resolution of 16 bits ($4000/2^{16} \approx 0.061$).

C. Data Representation

Before looking into the gyroscope fingerprinting technique, we first investigate the data representation format used by the hardware. We start by looking at the binary representation of the raw sensor data. In particular, we collect 3,918 raw gyroscope samples from an iPad Air 2. Each sample is a 3-tuple consisting of triaxial measurements; the native unit provided by the iOS SDK is radians per second. A visualization

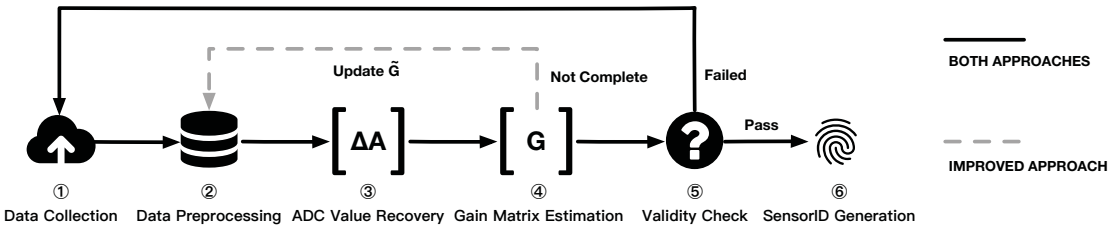


Fig. 3: General steps to recover the SENSORID

of the raw gyroscope data, in the IEEE 754 floating point representation, is shown in Fig. 2 (a).

Here, the colors show the various parts of the IEEE 754 representation: bit 1 (yellow) is the sign bit, bits 2–12 (green) are the exponent, and bits 13–64 (blue) are the significand. Overall, no obvious structure is observed in this form. However, if we convert the data into units of degrees per second and replot, then we obtain the results shown in Fig. 2 (b).

It is now obvious that the internal calculations have much less precision than is available in IEEE 754 representation and must be in units of degrees per second. We gain some more information if we convert the double precision numbers, in degrees per second (dps), into fixed-point Q32.32 form. Results are presented in Fig. 2 (c).

In Fig. 2 (c), the first 32 bits contain the integer part of the data (two's complement) and the last 32 bits show the fractional part. It is obvious that there are only 16 bits of data in the fractional part. The 16-bit resolution of gyroscope outputs is observed on all the iOS devices we have tested. There are a few possible reasons for this, but the simplest is that the value in the gain matrix $\mathbf{G}$ is stored as a signed integer with a resolution of 2^{-16} dps. After investigation, we find that every device that uses an M-series motion coprocessor, which was released by Apple in September 2013 with the iPhone 5S, shows this pattern. The purpose of the motion coprocessor is to offload the collection and processing of sensor data from the CPU. However, for older devices such as iPhone 4 and iPhone 4S, gain matrix values are stored with more precision and the calibration involves truncation down to 2^{-16} dps after the gain is applied. The complete set of devices that use the M-series motion coprocessor can be found online.[3] As discussed in §III-D, the lack of resolution in these devices allows us to precisely recover the gain matrix $\mathbf{G}$.

D. Fingerprinting from Mobile Apps

The general process to generate the SENSORID is illustrated in Fig. 3, which consists of six major steps: *Data Collection*, *Data Preprocessing*, *ADC Value Recovery*, *Gain Matrix Estimation*, *Validity Check*, and *SensorID Generation*. In the following of the section, we first present a basic approach to generate the gyroscope fingerprint (GYROID), which works well when the device is stationary. Then, we further propose an improved scheme that can reliably generate the GYROID even when the device is moving.

[3]https://en.wikipedia.org/wiki/Apple_motion_coprocessors

1) Basic Approach: We first consider the case that the device is stationary or moving slowly during sampling.

Data Collection. We collect a small number of samples from the gyroscope through a mobile app at the maximum sampling frequency. Empirically, we find 100 samples collected in less than 1 second is sufficient. We use $\mathbf{O} = [\mathbf{O}_0, \mathbf{O}_1, \cdots, \mathbf{O}_{n-1}]$ to denote the collected data, where $\mathbf{O}_i = [O_{i_x}, O_{i_y}, O_{i_z}]^T$ is a 3-by-1 vector.

Data Preprocessing. After collecting the data, we calculate $\Delta\mathbf{O}$ by differencing the consecutive outputs for all three axes. In other words, $\Delta\mathbf{O}$ is calculated by the following equation:

$$\Delta\mathbf{O} = [\mathbf{O}_1 - \mathbf{O}_0, \mathbf{O}_2 - \mathbf{O}_1, \cdots, \mathbf{O}_{n-1} - \mathbf{O}_{n-2}]$$

ADC Value Recovery. In this step, we aim to recover $\Delta\mathbf{A}$, which is the difference between consecutive ADC outputs. From Equation 3 we know that $\Delta\mathbf{A}$ can be calculated by:

$$\Delta\mathbf{A} = \mathbf{G}^{-1}\Delta\mathbf{O} \tag{5}$$

where $\mathbf{G}^{-1}$ is the inverse of the gain matrix $\mathbf{G}$. However, the value of $\mathbf{G}$ is unknown at the moment. Nevertheless, we can set $\mathbf{G}$ to the ideal value $\mathbf{G}_0 = F_G \cdot \mathbf{I}$, where F_G is the nominal gain of the gyroscope. Since the deterministic errors are comparatively small, $\mathbf{G}$ should be relatively close to $\mathbf{G}_0$ (§III-B). Since $\Delta\mathbf{A}$ only has integer values, we can estimate $\Delta\mathbf{A}$ by:

$$\widetilde{\Delta\mathbf{A}} = \mathrm{round}(\mathbf{G}_0^{-1}\Delta\mathbf{O}) \tag{6}$$

where the round function rounds each element to the nearest integer. However, since $\mathbf{G}_0$ is not equal to $\mathbf{G}$, the rounded value $\widetilde{\Delta\mathbf{A}}$ may not be the true value. Therefore, we further calculate the rounding error $\mathbf{E}^r_{\Delta\mathbf{A}} \in \mathbb{R}^{3\times(n-1)}$, which is defined as follows:

$$\mathbf{E}^r_{\Delta\mathbf{A}} = |\widetilde{\Delta\mathbf{A}} - \mathbf{G}_0^{-1}\Delta\mathbf{O}| \tag{7}$$

To ensure the estimated values are correct, we require that every value in $\mathbf{E}^r_{\Delta\mathbf{A}_i}$, which means the i-th column in $\mathbf{E}^r_{\Delta\mathbf{A}}$, be lower than a threshold Γ (e.g., 0.1). If not, we believe the rounding is ambiguous and remove both $\widetilde{\Delta\mathbf{A}}_i$ and $\Delta\mathbf{O}_i$ from the dataset. Once all ambiguous values are removed, we will regard the estimated $\widetilde{\Delta\mathbf{A}}$ as the true ADC value matrix $\Delta\mathbf{A}$.

Note that, this is only true if the device is stationary or moving slowly, in which case the absolute values of $\Delta\mathbf{A}$ are small. Otherwise, any rounding error could be accumulated and results in rounding to an incorrect integer value.

Gain Matrix Estimation. After recovering the ADC value matrix $\Delta\mathbf{A}$, we can estimate the gain matrix $\widetilde{\mathbf{G}}$ as follows:

$$\widetilde{\mathbf{G}} = \Delta\mathbf{O}/\Delta\mathbf{A}$$

Here, the matrix right division operation returns a least squares solution of $\mathbf{G}$ that minimizes the norm of the vector $\mathbf{G}\Delta\mathbf{A} - \Delta\mathbf{O}$.

Validity Check. To quantify the deviation between $\widetilde{\mathbf{G}}$ and the true value of $\mathbf{G}$, we define the estimation error $\mathbf{E}^e \in \mathbb{R}^{3\times 1}$ as follows:

$$\mathbf{E}^e = \mathrm{std}(\Delta\mathbf{O} - \widetilde{\mathbf{G}}\Delta\mathbf{A})$$

where std is the row-wise standard deviation function. If the estimation error is small (i.e., $\max(\mathbf{E}^e) < \Theta$), then $\widetilde{\mathbf{G}}$ should be close to the true gain matrix $\mathbf{G}$.

If the device uses an M-series motion coprocessor, we can implement another check to ensure the accuracy of our estimation and precisely recover the true gain matrix $\mathbf{G}$.

By way of an example, here is the $\widetilde{\mathbf{G}}$ that we estimated from an iPhone XS in the units of radians per second (rps):

$$\begin{bmatrix} 0.001068460229340 & -0.000009587379924 & -0.000002929477199 \\ 0.000002929477199 & 0.001073520235411 & 0.000005858954398 \\ -0.000001065264436 & -0.000006657902725 & 0.001069525493776 \end{bmatrix}$$

Recall that the gain matrix of the iPhone XS, which uses Apple M12 coprocessor, only has 2^{-16} resolution in the units of dps. Therefore we can convert from rds to dps and estimate the underlying integers:

$$\widetilde{\mathbf{G}} \cdot 2^{16} \cdot 180/\pi =$$

$$\begin{bmatrix} 4012.000000000001 & -35.999999999999318 & -10.999999999999677 \\ 11.000000000000174 & 4030.999999999999 & 21.999999999999631 \\ -3.999999999999980 & -25.000000000000011 & 4016.000000000000 \end{bmatrix}$$

In this case, these numbers are extremely close to whole integers, so these can be rounded to obtain the true gain matrix $\mathbf{G}$. To ensure the rounding is safe, we calculate the rounding error $\mathbf{E}^r_{\mathbf{G}} \in \mathbb{R}^{3\times 3}$ as follows:

$$\mathbf{E}^r_{\mathbf{G}} = |\widetilde{\mathbf{G}^d} \cdot 2^{16} - \mathrm{round}(\widetilde{\mathbf{G}^d} \cdot 2^{16})|$$

where $\widetilde{\mathbf{G}^d}$ means $\widetilde{\mathbf{G}}$ in the units of dps.

If the maximum value in $\mathbf{E}^r_{\mathbf{G}}$ is lower than a threshold Φ (e.g., 0.01), then we believe that the rounding is reliable. Otherwise, if any of these checks fail, then it is likely that the device was moving during data collection. For this basic approach, we need to repeatedly collect another batch of data until the estimation error and rounding error are small enough.

GyroID Generation. The generation of the GyroID can be categorized into two groups based on whether the device has an M-series coprocessor. If the device does have an M-series coprocessor, the GyroID is defined as follows:

$$\text{GyroID} = \mathrm{round}(\widetilde{\mathbf{G}^d} \cdot 2^{16}) \ - \ \mathrm{round}(\mathbf{G}^d_0 \cdot 2^{16}) \qquad (8)$$

Or in words, the GyroID is the gain matrix $\mathbf{G}$ after subtracting the nominal gain in units of 2^{-16} dps. For instance, the GyroID of the iPhone XS in previous example is:

$$\text{GyroID} = \begin{bmatrix} 14 & -36 & -11 \\ 11 & 33 & 22 \\ -4 & -25 & 18 \end{bmatrix}$$

§III-F covers the case when the device does not contain an M-series coprocessor.

Summary. The basic approach illustrates the general idea and procedure to generate the GyroID. Overall, the calculations are light-weight and are easy to implement. However, the basic approach requires the device to be stationary or moving slowly during measurement. To address this problem, we propose an improved approach which takes device movement into consideration.

2) Improved Approach: The drawback of the basic approach is that it may take a long time to generate the GyroID, because the approach will keep trying until the device is almost stationary. In Equation 6, we use a bootstrap value $\mathbf{G}_0 = F_G \cdot \mathbf{I}$ to estimate the value of $\Delta\mathbf{A}$. However, since $\mathbf{G}_0$ is not equal to $\mathbf{G}$, the rounded value $\widetilde{\Delta\mathbf{A}}$ may not be the actual value $\Delta\mathbf{A}$. In general, the difference between $\Delta\mathbf{O}$ and $\mathbf{G}_0 \cdot \Delta\mathbf{A}$ will increase as elements in $\Delta\mathbf{O}$ gets bigger, leading to incorrect rounding (i.e., $\widetilde{\Delta\mathbf{A}} \neq \Delta\mathbf{A}$). In the improved approach, we use the same processing steps as in §III-D1 except for *Data Preprocessing*, *ADC Value Recovery*, and *Gain Matrix Estimation*. The general idea is that, instead of feeding $\Delta\mathbf{O}$ directly into the algorithm (which might result in incorrect rounding), we update $\mathbf{G}$ iteratively using data with different ranges. The general steps of the improved approach are also illustrated in Fig. 3.

Data Preprocessing. In this step, we first generate more data from $\Delta\mathbf{O}$ with small values because smaller values are less likely to introduce rounding errors. To do so, we can sort elements in $\Delta\mathbf{O}$ and then take the difference between adjacent elements; the resulting values are then likely to be small. In more detail, suppose $\Delta\mathbf{O} = [\Delta\mathbf{O}_0, \Delta\mathbf{O}_1, \cdots, \Delta\mathbf{O}_{n-1}]$, where $\Delta\mathbf{O}_i = [\Delta O_{i_x}, \Delta O_{i_y}, \Delta O_{i_z}]^T$ is a 3-by-1 vector. We first sort $\Delta\mathbf{O}$ based on the value of ΔO_{i_x} into ascending order. Here, we use $[\Delta\mathbf{O}]_x$ to denote the sorted array. Similarly, we sort $\Delta\mathbf{O}$ by the value of ΔO_{i_y} and ΔO_{i_z}, and denote the results as $[\Delta\mathbf{O}]_y$ and $[\Delta\mathbf{O}]_z$, respectively. Then, we calculate $\Delta\Delta\mathbf{O}$ as follows:

$$\Delta\Delta\mathbf{O} = [\mathrm{diff}([\Delta\mathbf{O}]_x)\ \mathrm{diff}([\Delta\mathbf{O}]_y)\ \mathrm{diff}([\Delta\mathbf{O}]_z)] \qquad (9)$$

where the diff function differences consecutive column vectors in a matrix. For instance, $\mathrm{diff}(\Delta\mathbf{O})$ is calculated by:

$$\mathrm{diff}(\Delta\mathbf{O}) = [\Delta\mathbf{O}_1 - \Delta\mathbf{O}_0, \cdots, \Delta\mathbf{O}_{n-1} - \Delta\mathbf{O}_{n-2}]$$

By subtracting similar vectors, $\Delta\Delta\mathbf{O}$ contains more data with smaller values. From Equation 5 we have:

$$\mathbf{G}^{-1}\mathrm{diff}(\Delta\mathbf{O}) = \mathrm{diff}(\Delta\mathbf{A})$$

where values in $\mathrm{diff}(\Delta\mathbf{A})$ are all integers. Combined with Equation 9, it is clear that the result of $\mathbf{G}^{-1}\Delta\Delta\mathbf{O}$ should only contain integer values. Therefore, we can directly add $\Delta\Delta\mathbf{O}$ to the $\Delta\mathbf{O}$ dataset and our basic algorithm still applies (i.e., $\Delta\mathbf{O} \leftarrow [\Delta\mathbf{O}\ \Delta\Delta\mathbf{O}]$). This expansion can be implemented multiple times to generate more data with small values.

Then, we generate several batches of data from the expanded dataset based on the value range and update $\widetilde{\mathbf{G}}$

iteratively. In particular, each batch, $\Delta\mathbf{O}^{\epsilon_i}$, is a subset of $\Delta\mathbf{O}$ where the absolute value of all its elements is lower than a multiplication of the nominal gain. That is to say,

$$\Delta\mathbf{O}^{\epsilon_i} = \{\Delta\mathbf{O}_j \in \Delta\mathbf{O} \mid \max(|\Delta\mathbf{O}_j|) < (\epsilon_i + 0.5)F_G\}$$

where ϵ_i is the threshold for batch i. We start ϵ_i from 1 and double its value for each batch (i.e., $\epsilon_{i+1} = 2\epsilon_i$) until the batch is the same as $\Delta\mathbf{O}$. Then, we progressively feed $\Delta\mathbf{O}^{\epsilon_i}$ to the next step and update the value of $\mathbf{G}$ from the first batch to the last one.

ADC Value Recovery. In this step, instead of using $\mathbf{G}_0$ to estimate $\Delta\mathbf{A}$ and calculate the rounding error $\mathbf{E}^r_{\Delta\mathbf{A}}$, we instead use $\widetilde{\mathbf{G}}$ whose initial value is $\mathbf{G}_0$. In other words, Equation 6 and 7 should be updated to:

$$\widetilde{\Delta\mathbf{A}}^{\epsilon_i} = \text{round}(\widetilde{\mathbf{G}}^{-1}\Delta\mathbf{O}^{\epsilon_i})$$
$$\mathbf{E}^r_{\Delta\mathbf{A}^{\epsilon_i}} = |\widetilde{\Delta\mathbf{A}}^{\epsilon_i} - \widetilde{\mathbf{G}}^{-1}\Delta\mathbf{O}^{\epsilon_i}|$$

Gain Matrix Estimation. This step updates $\widetilde{\mathbf{G}}$ from each batch of data, $\Delta\mathbf{O}^{\epsilon_i}$. As seen in Fig. 3, one major difference between the improved approach and the basic approach is that the former may go back to the *Data Preprocessing* step after the *Gain Matrix Estimation*. In particular, after each update of $\widetilde{\mathbf{G}}$, the algorithm will check whether $\Delta\mathbf{O}^{\epsilon_i} = \Delta\mathbf{O}$ holds true. If true, it means we have processed all the output data, and we will pass the estimated $\widetilde{\mathbf{G}}$ to the *Validity Check* process. Otherwise, the algorithm will go back to the *Data Preprocessing* stage with an updated $\widetilde{\mathbf{G}}$, and a new batch of data will be processed with $\epsilon_{i+1} = 2\epsilon_i$.

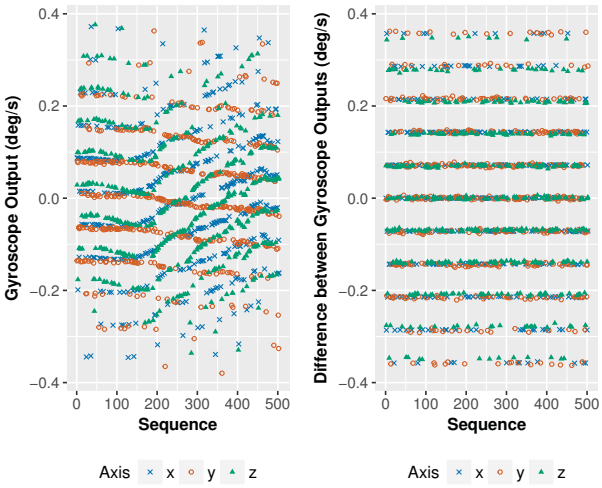
Summary. The improved approach updates the estimation of $\mathbf{G}$ iteratively from data within a small range to the whole dataset. By using the iteratively updated $\mathbf{G}$ to estimate ADC outputs, the improved approach reduces the error of each estimation, and thus, it is able to generate a GYROID even when the device is moving modestly. In general, the basic approach is useful to illustrate the idea. The improved approach introduces few additional computations and is much more reliable. Therefore, it is the preferred way to fingerprint devices in practical situations.

E. Fingerprinting from Mobile Websites

JavaScript also provides APIs for web developers to access the fused gyroscope data. Fig. 4 (a) presents the data collected from an iPad Air through mobile Safari. As shown in Fig. 4 (a), quantization in the fused data is still visible because the gyroscope ADC outputs are integers. However, there is a slowly varying continuous component added to the bias. The bias correction is likely to be a Kalman filter. Fig. 4 (b) shows that the bias part can be nearly eliminated by subtracting consecutive samples. Therefore, we can apply the same technique described in §III-D to recover the gain matrix.

F. Practical Calibration Fingerprinting Attacks

To launch a calibration fingerprinting attack, an adversary can collect gyroscope samples from any device using an app written by the attacker or that visits any website under the attacker's control. The attacker can then generate a device fingerprint (i.e., GYROID) from the samples and store it in a database. Then, the adversary can query the database to determine when a particular physical device uses a particular app or visits a particular website. The details of the generation and query of the GYROID in the database differ depending on the collecting source (app or web) and device model.



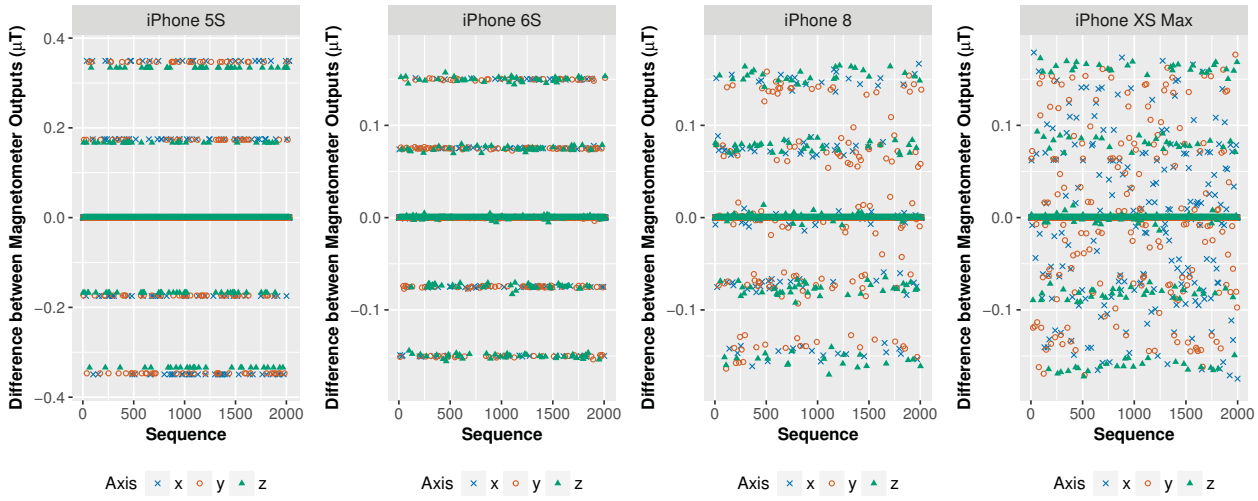
(a) Fused gyroscope data (b) After differencing

Fig. 4: Gyroscope data collected via JavaScript (iPad Air)

There are two cases to consider if the GYROID is generated from raw sensor data. First, if the device contains an M-series motion coprocessor, the adversary can apply the approach presented in §III-D2 to generate the GYROID; this GyroID is exact and uniquely identifies the device. Otherwise, if the device does not have an M-series coprocessor (e.g., iPhone 4, iPhone 4S, and iPhone 5), adversaries have two options to determine whether two GYROID entries in the database represent the same physical device.

- Option 1 (Clustering): the adversary can directly use the estimated gain matrix $\widetilde{\mathbf{G}}$ as the GYROID and store it in a database. For every new device with the same model, the attacker can calculate its $\widetilde{\mathbf{G}}$ and compare the Euclidean distance between its $\widetilde{\mathbf{G}}$ and the ones in the databases. If they are close, then they can know they came from the same device.
- Option 2 (Rounding): For devices without an M-series coprocessor, we find that if we use Equation 8 to generate the GYROID, this deviates from the true GYROID by at most ± 1 for each of the 9 values in the gain matrix. By way of an example, the estimated gain matrix of an iPhone 5 is given in Appendix B. Therefore, an adversary can simply store the estimated gain matrix and perform a fuzzy query (i.e. accept a ± 1 fluctuation for each element). Note that this option provides less entropy compared with Option 1.

The bias correction applied to the JavaScript API means that



(a) Type I (b) Type II (c) Type III (d) Type IV

Fig. 5: Consecutive differences between raw magnetometer measurements for different devices

an attacker cannot achieve the same precision as is possible with raw gyroscope data. In this case, the adversary also has two choices:

- Option 1 (Clustering): the adversary can collect multiple batches of gyroscope samples from the device and calculate the estimated gain matrix $\widetilde{\mathbf{G}}$ for each batch. Then, they can use the set of $\widetilde{\mathbf{G}}$ as the device fingerprint and compare it with records in the database using a clustering algorithm. If the set of $\widetilde{\mathbf{G}}$ shows the same pattern as the one in the database, then it is likely that they are the same device (the entropy of $\mathbf{G}$ is discussed in §V-A).
- Option 2 (Rounding): the adversary can round values in $\widetilde{\mathbf{G}}$ to the nearest integer in units of 2^{-16} dps. Because there is a varying bias drift, the rounded $\widetilde{\mathbf{G}}$ may not be equal to the rounded $\mathbf{G}$. However, the adversary can implement a grid search for each element in the rounded $\widetilde{\mathbf{G}}$ by increasing or decreasing its value by a small integer. Then, adversaries can update $\widetilde{\mathbf{G}}$ by the combination that produces the minimum estimation error $\mathbf{E}^e$, and the results will be passed to the *Validity Check* process for further checking. In our experiments, we found that the check will only pass if the device is almost stationary (e.g., on a desk or held in hand) during sampling, and the difference between values in the estimated $\widetilde{\mathbf{G}}$ and that in the rounded $\mathbf{G}$ is in ± 1. If the device is under intense movement, the algorithm will collect another batch of data and keep trying until the device is nearly stationary.

G. Fingerprinting Other Motion Sensors

We also investigated whether our approach can be applied to recover the gain matrix of the accelerometer and magnetometer. As a result, we found that our approach does not directly apply to the accelerometer of iOS devices. However, we discovered that the accelerometer in Google Pixel 2/3 devices can be fingerprinted (§V-D). In addition, the magnetometer in iOS devices is also fingerprintable. Similar to the gyroscope, the raw readings from the magnetometer only have a resolution of 2^{-16} μT (microtesla). As shown in Fig. 5, after subtracting consecutive raw magnetometer measurements for every device model in our dataset, we observed four types of pattern:

- Type I (different sensitivity, negligible fluctuation): Type I devices have a different but similar sensitivity for each magnetometer axis. As shown in Fig. 5 (a), the sensitivity of z axis is clearly lower than that of the x and y axes. The fluctuation within each cluster is negligible. In this case, the nonorthogonality matrix $\mathbf{N}$ in Equation 1 is nearly an identity matrix. However, the scale factor matrix $\mathbf{S}$ may be different across devices, which could be used as a fingerprint.
- Type II (fixed sensitivity, moderate fluctuation): Type II devices all have a same fixed sensitivity (4915×2^{-16} μT) for every axis. Therefore, the scale factor matrix $\mathbf{S}$ does not provide us with any entropy. However, there is a moderate fluctuation within each cluster, which indicates the existence of a non-identity nonorthogonality matrix $\mathbf{N}$. The nonorthogonality matrix $\mathbf{N}$ may be different across devices, which could be used as a fingerprint.
- Type III (different sensitivity, moderate fluctuation): Type III devices have different but similar sensitivity for each axis and there is a moderate fluctuation within each cluster. Nevertheless, the quantization of the data is still evident in this case.
- Type IV (different sensitivity, intense fluctuation): Type IV devices show an intense fluctuation on the magnetometer output. In this case, the quantization of the data is not as evident as in the other cases.

We summarize the magnetometer type of different iOS

TABLE II: Magnetometer type of different iOS device models

Type	Model	Nominal Gain
Type I	iPhone 4S/5/5C/5S/6/6 Plus All iPad models	0.35/0.28/0.17*
Type II	iPhone 6S/6S Plus/7/7 Plus/SE	0.075
Type III	iPhone 8/8 Plus	0.075
Type IV	iPhone X/XS/XS Max	0.075

* Type I devices have three possible nominal gain values.

TABLE III: MAGID Fingerprinting Result

Type	# Devices	Group Size	# Groups
Type I	213	1 2	193 10
Type II	434	1	434
Type III	96	1	96
Type IV	52	1	52

device models and their estimated nominal gain in Table II. Overall, the observation of the four patterns reveals the different underlying calibration procedures. For all four types of devices, we can use the same approach described in §III-D to obtain the magnetometer fingerprint (i.e., MAGID). Although the gain matrix of the magnetometer does not appear to be stored at 2^{-16} μT resolution, adversaries can use the same techniques discussed in §III-F to launch an attack by either clustering or rounding.

Compared with the gyroscope, the raw magnetometer data is not currently accessible in major browsers. Nevertheless, the MAGID provides additional entropy to the GYROID. Thus, we can combine them together when analyzing apps as a finer-grained fingerprint. In this paper, we define the SENSORID as a combination of distinctive sensor calibration fingerprints. In the case of iOS devices, the SENSORID includes both the GYROID and MAGID.

IV. EVALUATION

We developed both a website[4] and an iOS app to collect sensor data. The iOS app collects raw data from the gyroscope and magnetometer at 200 Hz and does not ask users to put the device in any particular position. Additionally, it uses an embedded webview to collect fused gyroscope data and a standard iOS API call to record the current device model (e.g., "iPhone10,3").

The website is only able to collect fused gyroscope data and the device model. For both the app and website, we use the Fingerprintjs2 [14] library in the default configuration to generate a browser fingerprint for evaluation purposes. In addition to volunteers, we recruited participants from both the Amazon Mechanical Turk[5] and Prolific[6] to download the app and contribute sensor data. The public data collection exercise has been approved by the ethics committee of the Department of Computer Science and Technology in the University of Cambridge.

The SENSORID app has collected data from 795 unique iOS devices. In particular, 761 of them contain an M-series motion coprocessor. In addition, the website has collected fused data from another 75 devices. Some users chose to participate in this study multiple times. Thus, there might be more than one record for each unique device. For both the app and the website, we ask users to tell us whether they have submitted the data from this device before.

[4]https://sensorid.cl.cam.ac.uk/
[5]https://www.mturk.com
[6]https://prolific.ac

A. Results

Using the raw gyroscope data collected from the 761 iOS devices with an M-series coprocessor, we are able to recover the gain matrix of their gyroscope exactly and generate the GYROID. For the other 34 devices that do not contain an M-series coprocessor, we use the rounding option in §III-F to generate the GYROID due to the small sample size. Based on the GYROID, we successfully identify multiple records that are submitted by the same device. This is confirmed by user-supplied data about whether they have submitted samples from this device before and the device IP address when they submit. The GYROID of each device is distinct.

Since the website only collects fused gyroscope data, we choose the rounding option in §III-F to generate the GYROID. Then, we compare it with the GYROID of the 795 devices that we recovered from the raw data. As a result, we identify 3 devices that submitted through both the website and the app. The app also collects fused sensor data from the built-in webview. For this data, we use the clustering approach to generate a group of gain matrix estimates. Then, we apply the Multivariate ANalysis Of VAriance (MANOVA) technique to analyze these estimates and successfully identify all 795 unique devices in the dataset. In particular, we also identify 6 devices that submitted multiple times through the app. The results are the same as we obtained from the raw data.

In addition, we apply the improved approach to fingerprint the magnetometer with the rounding option. After generating the MAGID, we group devices by their MAGID and present the results in Table III. In the table, the group size records the number of different devices sharing the same MAGID. Therefore, a group of size 1 means the device has a unique MAGID in our dataset. In addition, Table III categorizes the results based on the device type and presents the number of unique devices in each category. As shown in the table, some Type I devices share the same MAGID. It reveals that Type I devices have a higher chance of collision on MAGID than others. The reason is that the entropy of the MAGID for Type I devices is only provided by the scale matrix (i.e., main-diagonal elements in MAGID). Nevertheless, the MAGID is orthogonal to the GYROID, and thus, they can be combined together to provide additional entropy.

We also compare the GYROID and MAGID with the Fingerprintjs2 fingerprint, which utilizes font detection, canvas

TABLE IV: Comparison of Different Device Fingerprints

Fingerprint	GyroID	MagID		Fingerprintjs2																
# Devices	870	795		870																
Group Size	1	1	2	1	2	3	4	5	6	7	9	10	11	13	14	19	22	28	36	45
# Groups	870	775	10	391	43	22	12	2	4	2	1	1	2	2	1	1	1	1	1	1

fingerprint, WebGL fingerprint, etc. Table IV shows that both the GyroID and MagID provide more entropy than traditional browser fingerprinting techniques. While GyroID is unique for every device in our dataset, 45 out of 135 iPhone 7 devices have the same Fingerprintjs2 fingerprint.

B. Proof of Concept

We have developed a proof of concept app for iOS devices with an M-series motion coprocessor. The app implements the improved approach (§III-D2) to generate the GyroID of the test device. The code is written in Swift 4.1 and compiled by XCode 9.4.1. The screenshots of the app are shown in Fig. 8 in Appendix C. After the user clicks the "Generate GyroID" button (Fig. 8 (a)), the app collects 100 raw gyroscope samples and attempts to generate the GyroID. If it fails (due to intense shaking of the phone) the app automatically collects another 100 raw samples and repeats the process. Fig. 8 (b) shows the result from a handheld iPhone XS running iOS 12.1.1. As seen in the Figure, it takes about 0.5 seconds to collect 100 gyroscope samples and the whole data processing procedure finishes within 0.01 seconds. We have also tested shaking the device during data collection. Vigorous movement during extraction may require additional samples, but the task nevertheless completes within a few hundred samples and takes a few seconds. Regardless of the device movement and CPU load, the generated GyroID always stays the same.

V. Discussion

In this section, we discuss some possible threats that could impair the validity of this research.

A. *Is* SensorID *unique?*

The first question to ask is whether SensorID has enough entropy to be used as a device fingerprint. In other words, is SensorID likely to be unique? To answer this question, we studied the GyroID of all iOS devices with an estimated nominal gain of 61 mdps. Device models included in this category can be found in Table I. We chose this category for two reasons. First, all device models in this category are modern devices which contain an M-series motion coprocessor and this makes possible to extract their exact gain matrix. Second, devices with different default gain may have a different GyroID distribution, so we select the larger size group, which contains 693 devices in total. Fig. 9 in Appendix D presents the distribution of each element in GyroID for the 693 devices. For simplicity, we denote the GyroID as $\mathbf{D} \in \mathbb{Z}^{3\times3}$ in the following analysis.

Normality Analysis. Through analyzing the GyroID of the 693 devices, we find that each element in $\mathbf{D}$ appears to be normally distributed. To test for normality, we applied both the Kolmogorov-Smirnov test and the Shapiro-Wilk test of normality for each element in $\mathbf{D}$. Results show that elements that are not in the the main diagonal of $\mathbf{D}$, including $\mathbf{D}_{12}, \mathbf{D}_{13}, \mathbf{D}_{21}, \mathbf{D}_{23}, \mathbf{D}_{31}, \mathbf{D}_{32}$, have a strong normality. Elements in the main diagonal ($\mathbf{D}_{11}$, $\mathbf{D}_{22}$, $\mathbf{D}_{33}$) are rejected by both tests at the 0.05 significance level. Fig. 10 in Appendix D presents the Q-Q plot of each $\mathbf{D}_{ij} \in \mathbf{D}$ that gives an intuitive view of the normality of each element.

The result implies that the non-diagonal elements are normally distributed, and we can use this to estimate their entropy. In addition, it shows that we may need finer-grained analysis to calculate the entropy of diagonal elements. The elements in the main diagonal of $\mathbf{D}$ are strongly affected by the default gain. Although we only choose device models with an estimated gain of 61 mdps, the actual default gain of each device model may deviate from our estimation. Therefore, we need to do per-device type analysis for each element in the main diagonal. When we run a normality test on data from each per-device type separately we find that the diagonal elements appear to be normally distributed.

Correlation Analysis. We also investigate whether elements in $\mathbf{D}$ are strongly correlated with each other. For this purpose, we run the Pearson correlation test on each $\mathbf{D}_{ij}$ and find that $\mathbf{D}_{12}$ and $\mathbf{D}_{13}$ are correlated with $\mathbf{D}_{21}$ and $\mathbf{D}_{31}$ at the 0.01 significance level, respectively. Based on this result, we regard both $\mathbf{D}_{21}$ and $\mathbf{D}_{31}$ as dependent variables and do not include them in our entropy calculation. An intuitive view of the correlation between different elements in $\mathbf{D}$ is presented in Fig. 11 in Appendix D.

Entropy Calculation. We first calculate the entropy of non-diagonal elements in $\mathbf{D}$, excluding $\mathbf{D}_{21}$ and $\mathbf{D}_{31}$. For each non-diagonal element, we estimate the parameters of the normal distribution, including the mean μ and standard deviation σ, from the dataset. Technically, it is not a strict normal distribution since each element can only be an integer. Nevertheless, the quantization is a result of rounding. Thus, we can still apply the concept of normal distribution to estimate the entropy.

In general, the entropy of a discrete random variable X, which is denoted as $\mathrm{H}(X)$, can be calculated by:

$$\mathrm{H}(X) = -\sum_{x_i \in X} \mathrm{P}(x_i) \log_2 \mathrm{P}(x_i) \tag{10}$$

where $\mathrm{P}(x_i)$ is the probability of X being equal to x_i. In our case, we regard the element $\mathbf{D}_{ij}$ as the variable X. Then, we have $x_i \in \{-65535, \ldots, 65535\}$ because of the 16-bit

TABLE V: Comparison of Different Device Fingerprints

Variable	$\mathbf{D}_{11}$	$\mathbf{D}_{12}$	$\mathbf{D}_{13}$	$\mathbf{D}_{22}$	$\mathbf{D}_{23}$	$\mathbf{D}_{32}$	$\mathbf{D}_{33}$	$\mathbf{D}$
Entropy (bits)	5.0	6.7	7.2	5.0	6.5	6.5	5.4	42.3

resolution. Suppose $X \sim \mathcal{N}(\mu, \sigma^2)$ with the density function $f(x)$, then we can calculate $\mathrm{P}(x_i)$ as follows:

$$\mathrm{P}(x_i) = \begin{cases} \int_{x_i-0.5}^{x_i+0.5} f(x)\ dx, & \text{if } x_i \in (-65535, 65535) \\ \int_{-\infty}^{-65534.5} f(x)\ dx, & \text{if } x_i = -65535 \\ \int_{65534.5}^{+\infty} f(x)\ dx, & \text{if } x_i = 65535 \end{cases} \tag{11}$$

By this equation, we calculate the entropy of $\mathbf{D}_{12}$, $\mathbf{D}_{13}$, $\mathbf{D}_{23}$, and $\mathbf{D}_{32}$. Results are presented in Table V.

For elements in the main diagonal (i.e., $\mathbf{D}_{11}$, $\mathbf{D}_{22}$, and $\mathbf{D}_{33}$), we need to calculate their entropy on a per device type basis. Here, we use the iPhone 6S as an example to calculate the GYROID entropy because it is the most popular device model in our dataset (127 devices). Similar to the entropy calculation for non-diagonal elements, we estimate the parameters of normal distribution for diagonal elements from the 127 GYROID matrices. Then, we apply both Equation 10 and 11 to calculate the entropy. Table V depicts the GYROID entropy we estimated for the iPhone 6S. As seen in Table V, the GYROID for the iPhone 6S has about 42 bits ($\approx 4.4 \times 10^{12}$ dits) of entropy. By the same analysis, we estimate the entropy of the MAGID for the iPhone 6S. If adversaries launch the attack using the rounding option (§III-F), each element could have ± 1 uncertainty. In this case, we estimate that the MAGID contains about 25 bits of entropy. The MAGID should have more entropy if adversaries choose the clustering option. Thus, the 25 bits of entropy should be regarded as a lower bound for the MAGID. We observe no evidence of strong correlation between the MAGID and GYROID. Therefore, we estimate the SENSORID for the iPhone 6S has around 67 bits of entropy.

Uniqueness Analysis. There were 728M active iPhones worldwide in April 2017 and the iPhone 6S devices accounted for 18% of them [15]. Therefore, there were around 131M iPhone 6S devices. From the birthday problem, we know that the chance of two iPhone 6S devices having the same SENSORID is around 0.0058%, suggesting it is a globally unique device fingerprint. In addition, the SENSORID is orthogonal to other fingerprinting techniques. Therefore, adversaries can combine the SENSORID with other metadata (e.g., system language) or other fingerprinting techniques (e.g., canvas fingerprinting) to further increase the fingerprint entropy.

B. Is SENSORID correlated with the manufacturing batch?

To answer this question, we first study the correlation between the SENSORID and the country of the device, which is inferred from the IP address when a user submits data. We do not find any evidence of strong correction at the 0.05 significance level.

In addition, we collected gyroscope data from 25 iOS devices in an Apple Store. Some of these devices have similar serial numbers, which suggests they may come from the same manufacturing batch. However, the GYROID of these devices differs significantly. Furthermore, there is no significant difference in the GYROID distribution for devices from the Apple Store and for devices that we collect otherwise.

C. Will SENSORID ever change?

We have not observed any change in the SENSORID of our test devices in the past half year. Our dataset includes devices running iOS 9/10/11/12. We have tested compass calibration, factory reset, and updating iOS (including the latest iOS 12); the SENSORID always stays the same. We have also tried measuring the sensor data at different locations and under different temperatures; we confirm that these factors do not change the SENSORID either.

D. Factory calibration in Android devices

We have observed that the scale factors of the accelerometer on Google Pixel 2 and Pixel 3 devices are different for each axis (similar to Fig. 5 (a)), which suggest some form of factory calibration has been applied. For Pixel 3, the factory calibration values are stored in local files when the system is first booted. If the device is rooted, the calibration values can be directly read from these files. We use a similar approach to estimate the gain matrix of the accelerometer. The gain matrices we estimated from the sensor outputs of two Pixel 3 devices match the values stored in these local files and they are distinct. We only have access to a few Pixel devices, and therefore we are unable to perform the same analysis as we have done on iOS devices to determine whether the fingerprint we obtain for Pixel devices is globally unique or not. The IMU in other Android devices is also likely factory calibrated but the calibration is typically restricted to offsets (i.e., bias compensation). Our approach targets the gain matrix and cannot recover bias compensation.

E. Impact and coordinated disclosure

Overall, all iOS devices that have a gyroscope or magnetometer can be fingerprinted by this approach, including the latest iPhone XS and iPhone XS Max. A SENSORID can be generated by both apps and mobile websites and requires no user interaction. Both mainstream iOS browsers (Safari, Chrome, Firefox, and Opera) and privacy-enhanced browsers (Brave and Firefox Focus) are vulnerable to this calibration-based fingerprinting attack, even with the fingerprinting protection mode turned on. A recent study shows that motion sensor data is accessed by 2,653 of the Alexa top 100K websites, including more than 100 websites exfiltrating motion sensor data to remote servers [16]. This is troublesome since it is likely that the SENSORID can be calculated with exfiltrated data, allowing retrospective device fingerprinting.

We followed a coordinated disclosure procedure and reported this vulnerability to Apple on 3rd August 2018. On iOS

12.2, Apple adopted our suggestion (§VI) and added random noise to the ADC outputs. Apple also removed access to motion sensors from Mobile Safari by default. This vulnerability is disclosed to Google on 10th December 2018. Google has contacted us and is investigating this issue.

VI. Countermeasures

As shown in §IV and §V, the SensorID is a reliable device fingerprint that is easy and efficient to generate. In addition, this technique does not require any permission from the user and it works on both mobile apps and websites. As far as we know, the app review process currently enforced by Apple does not check the usage of sensor-related APIs. Therefore, it poses a big threat to user privacy.

To address this problem, manufacturers could regularly update the gain matrix through recalibration, either remotely or locally. However, this approach is not recommended. The gain matrix after recalibration is likely to be close to (or the same as) the original one, which may still give adversaries enough entropy to fingerprint the device. Furthermore, adversaries can still exploit the SensorID to fingerprint devices during the recalibrate-free time period. Apple could introduce runtime permissions for sensor APIs. However, existing work has shown that users tend to provide blind authorization to permission requests [17]. Additionally, it does not prevent apps that require access to motion sensors from fingerprinting users. Apple could also stop factory calibrating motion sensors in future iOS devices, but this will reduce sensor data quality and impair device usability; this also does not stop adversaries from fingerprinting existing iOS devices.

Here, we suggest two easier and more reliable countermeasures. Recall that our approach to recover the gain matrix is based on the fact that the values in the ADC output, $\mathbf{A}$, are all integers. This property allows us to recover the values of $\Delta\mathbf{A}$ using Equation 6. However, if we add random noise $\epsilon \in \mathbb{R}^{3\times 1}$, from the uniform distribution in the range $[-0.5, 0.5]$, to each ADC output $\mathbf{A}$. Then we have:

$$\mathbf{O} = \mathbf{G}(\mathbf{A} + \epsilon) + \mathbf{B} \tag{12}$$

It is clear to see that the application of random noise with distribution $\epsilon \sim U(-0.5, 0.5)$ followed by a truncation of the output down to 16-bit resolution means there is no easy way to recover the values of $\Delta\mathbf{A}$. Therefore, attackers can no longer generate the SensorID using the approach described in this paper. The cost of adding ϵ is negligible in most cases, but it offers significant advantages in terms of user privacy.

Alternatively, we could round the factory calibrated sensor output to the nearest multiple of the nominal gain to prevent recovering the gain matrix. This approach is more practical to apply and it does not require knowing the gain matrix in advance. Therefore, mobile browsers can adopt this approach to protect user privacy.

In addition, the World Wide Web Consortium (W3C) made a candidate recommendation of the generic sensor API, which allows web developers to access the magnetometer readings and possibly raw motion sensor data [18]. We believe this is detrimental to user privacy if a countermeasure is not deployed. Therefore, we suggest both the standards organization (e.g., W3C) and browser vendors treat the sensor-related API carefully and understand the privacy implications before adopting it.

VII. Related Work

A. Traditional Device Fingerprinting Techniques

Device fingerprinting is an important way for app developers and advertisers to track their users. The IP address is one of the earliest identifiers used to fingerprint devices. However, the adoption of dynamic IP allocation and Network Address Translation (NAT), particularly for home PCs and mobile devices, has greatly reduced the effectiveness of this approach. Cookies are also commonly used to track users across websites. However, cookies are stored locally and can be changed by users at any time. In fact, many privacy-focused browsers, such as Brave, by default block all third-party cookies. In addition, regulations in the US and Europe require websites to obtain user-permission before using cookies, which also decreases the usability of this approach [19].

A variety of IDs in the device can be used as fingerprints, including the IMEI, UDID, and MAC address of hardware modules. A study in 2011 showed that these identifiers were widely used in mobile apps [20]. However, both Apple and Google have adopted more stringent privacy policies to prevent developers from accessing these unique IDs. Additionally, many information flow tracking systems, such as TaintDroid [21] and Panorama [22], can capture these malicious behaviors and report them to users.

B. Passive Device Fingerprinting

Passive device fingerprinting is the action of characterizing a target device by observing its network traffic. It analyzes the captured data to reveal fingerprintable patterns (e.g., the software, operating system, or hardware components). Since passive fingerprinting only relies on network traffic, it is compatible with more devices, difficult to discover, and can track users across different browsers.

In general, most passive fingerprinting techniques rely on machine learning models to differentiate devices. Uluagac *et al.* applied Artificial Neural Networks (ANNs) to classify devices based on the time-variant behavior in their traffic [23]. Neumann *et al.* evaluated several features extracted from network traffic and found that the frame inter-arrival time, which is correlated with the hardware status and installed applications, is the most effective feature for device fingerprinting [24]. Machine learning approaches usually require more computing resources and a large amount of data for training. Thus, passive fingerprinting techniques usually have a longer response time than active fingerprinting techniques.

C. Active Device Fingerprinting

Active fingerprinting techniques deploy embedded code to actively gather information about a device and use these characteristics to make a distinction between different devices.

For example, Fingerprintjs2 [14] is a popular browser fingerprinting library that utilizes the characteristics of a browser, including the User-Agent (UA), version, plugins, font, and canvas. Apple has realized the risk of browser fingerprinting. From Mac OS Mojave, Safari scrubs most distinctive browser data, exposing only generic configuration information and default fonts [25]. The information about the operating system (e.g., version and root permission) and system configurations (e.g., network and flash configurations) can also be used to identify devices. Although this information cannot uniquely identify a device, it can be combined with other features from browsers and embedded hardware to increase precision.

D. Hardware Fingerprinting

In general, hardware imperfections are inevitable during manufacture, which implies the existence of fingerprints. Hardware modules that have been studied for fingerprinting purposes include: WiFi chipsets [26], [27], motion sensors [3], [28], camera [29], [30], and microphones [31].

For smartphones, hardware fingerprints are the most consistent identifier among the active fingerprinting sources. In general, it is difficult to replace the embedded hardware. Some embedded hardware, such as motion sensors, can be accessed by both JavaScript running in a web browser and by mobile apps installed on the device. In addition, accessing motion sensor data does not require any permission from users.

Existing hardware fingerprinting techniques are mostly based on machine learning approaches. Fridrich showed that the Photo-Response Non-Uniformity (PRNU) noise of imaging sensors can be estimated and used as a camera fingerprint [29]. Bojinov *et al.* demonstrated it is possible to fingerprint both the speakerphone-microphone system and the accelerometer using typical clustering approaches [28]. However, they only correctly identified 53% of the devices in their dataset even after integrating the UA string into their model. Das *et al.* applied several supervised machine learning models to make a distinction between devices based on the gyroscope and accelerometer readings [32]. To increase the accuracy, they used inaudible sound to stimulate the motion sensors. As a countermeasure, they suggested that manufacturers should perform better calibration of the motion sensors. However, they did not realize that the calibration process could leak information if not properly implemented. More recently, they further improved its accuracy by introducing a voting scheme among different classifiers [4]. Nevertheless, their approach requires a lot of computing resources, which cannot be implemented locally on the device. Even then, their approach achieved less than 60% F_1 score in an open-world setting when devices were held in hand. They also applied their approach to make a distinction between 85 iPhone 6 devices. When devices were held in hand, only 60% of these devices produced unique fingerprints, which, by reference to the birthday problem, indicates that their approach provides around 13 bits of entropy. Based on the motion sensor data that they collected through JavaScript, we correctly identify all iOS devices in the dataset based on the calibration behavior without knowing the device model in advance. Most recently, Das *et al.* studied the sensor API usage in popular websites [16]. They showed that 2,653 of the Alexa top 100K websites accessed motion sensor data and 63% of the scripts for accessing motion sensors also engaged in browser fingerprinting. Although the prior art has realized the idiosyncrasies across different sensors, none of them, to the best of our knowledge, has exploited the calibration information as a fingerprint. This paper fills this gap and shows that the calibration matrix of motion sensors can be utilized as a globally unique fingerprint for iOS devices.

VIII. Conclusion

In this paper we introduced the calibration fingerprinting attack: a new method of fingerprinting devices with embedded sensors by careful analysis of the sensor output alone. We demonstrated the effectiveness of this attack on iOS devices and found the lack of precision in the M-series co-processor helps the generation of such a fingerprint. Our attack is easy to conduct by a website or an app in under 1 second, requires no special permissions, does not require user interaction, and is computationally efficient. Our attack can also be applied retrospectively to a historic archive of sensor data. Using the iPhone 6S as an example, we showed that the GyroID contains about 42 bits of entropy and the MagID provides an additional 25 bits of entropy. Furthermore, we demonstrated that the combination of the MagID and GyroID – the SensorID – is globally unique for the iPhone 6S, does not change on factory reset or after a software update. In addition to iOS devices, we find that Google Pixel 2 and Pixel 3 can also be fingerprinted by our attack. To prevent this fingerprinting attack, we recommend the addition of uniformly distributed random noise to the output of the sensor before any calibration is applied or round the calibrated sensor output to the nearest multiple of nominal gain.

The idea of a calibration fingerprint attack is widely applicable. Although this paper mainly targets the gyroscope and magnetometer found in iOS devices, we anticipate calibration information used in other embedded sensors may also be recovered and used as a fingerprint, and therefore we expect future research will successfully perform calibration fingerprinting attacks on other types of sensor.

Acknowledgment

Jiexin Zhang is supported by the China Scholarship Council. Alastair R. Beresford is partly supported by The Boeing Company and EPSRC under Grant No.: EP/M020320/1. We thank Nokia Bell Labs for supporting this work and valuable discussions. The opinions, findings, and conclusions or recommendations expressed are those of the authors and do not necessarily reflect those of the funders. We thank Stephan A. Kollmann, Diana A. Vasile, Ricardo Mendes, Andrew Rice, and Amanda Prorok for helpful discussion and insight. We also thank our shepherd Adam Bates and anonymous reviewers for their feedback on the paper.

REFERENCES

[1] comScore. (2017) 2017 u.s. cross-platform future in focus. [Online]. Available: https://www.comscore.com/Insights/Presentations-and-Whitepapers/2017/2017-US-Cross-Platform-Future-in-Focus

[2] J. P. Achara, "Unveiling and controlling online tracking," Ph.D. dissertation, Université Grenoble-Alpes, 2016.

[3] S. Dey, N. Roy, W. Xu, R. R. Choudhury, and S. Nelakuditi, "Accelprint: imperfections of accelerometers make smartphones trackable," in *Proceedings of the 2014 Network and Distributed System Security (NDSS) Symposium*, 2014.

[4] A. Das, N. Borisov, and E. Chou, "Every move you make: exploring practical issues in smartphone motion sensor fingerprinting and countermeasures," *Proceedings on Privacy Enhancing Technologies Symposium (PETS)*, vol. 2018, no. 1, pp. 88–108, 2018.

[5] S. Poddar, V. Kumar, and A. Kumar, "A comprehensive overview of inertial sensor calibration techniques," *Journal of Dynamic Systems, Measurement, and Control*, vol. 139, no. 1, p. 011006, 2017.

[6] A. Grammenos, C. Mascolo, and J. Crowcroft, "You are sensing, but are you biased? a user unaided sensor calibration approach for mobile sensing," *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies (IMWUT)*, vol. 2, no. 1, p. 11, 2018.

[7] D. Tedaldi, "Imu calibration without mechanical equipment," Ph.D. dissertation, University of Padova, 2013.

[8] T. Michel, P. Genevès, H. Fourati, and N. Layaïda, "On attitude estimation with smartphones," in *Pervasive Computing and Communications (PerCom), 2017 IEEE International Conference on*. IEEE, 2017, pp. 267–275.

[9] D. Tedaldi, A. Pretto, and E. Menegatti, "A robust and easy to implement method for imu calibration without external equipments," in *2014 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2014, pp. 3042–3049.

[10] W. Ren, T. Zhang, H. Zhang, L. Wang, Y. Zhou, M. Luan, H. Liu, and J. Shi, "A research on calibration of low-precision mems inertial sensors," in *2013 25th Chinese Control and Decision Conference (CCDC)*. IEEE, 2013, pp. 3243–3247.

[11] I. Frosio, F. Pedersini, and N. A. Borghese, "Autocalibration of mems accelerometers," *IEEE Transactions on Instrumentation and Measurement*, vol. 58, no. 6, pp. 2034–2041, 2009.

[12] STMicroelectronics. (2010) L3g4200d: three axis digital output gyroscope. [Online]. Available: https://www.elecrow.com/download/L3G4200_AN3393.pdf

[13] ——. (2009) Lis331dlh: Mems digital output motion sensor. [Online]. Available: http://www.st.com/resource/en/datasheet/lis331dlh.pdf

[14] Valve. (2018) Modern & flexible browser fingerprinting library. [Online]. Available: https://github.com/Valve/fingerprintjs2

[15] J. Dunn. (2018) It looks like apple will have plenty of iphone owners that could use an upgrade this holiday season. [Online]. Available: http://uk.businessinsider.com/apple-iphone-most-popular-model-newzoo-chart-2017-7

[16] A. Das, G. Acar, N. Borisov, and A. Pradeep, "The web's sixth sense: a study of scripts accessing smartphone sensors," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2018.

[17] A. P. Felt, E. Ha, S. Egelman, A. Haney, E. Chin, and D. Wagner, "Android permissions: user attention, comprehension, and behavior," in *Proceedings of the Eighth Symposium on Usable Privacy and Security*. ACM, 2012, p. 3.

[18] W3C. (2018) Generic sensor api. [Online]. Available: https://www.w3.org/TR/generic-sensor

[19] Wikipedia. (2002) Privacy and electronic communications directive 2002. [Online]. Available: https://en.wikipedia.org/wiki/Privacy_and_Electronic_Communications_Directive_2002

[20] W. Enck, D. Octeau, P. D. McDaniel, and S. Chaudhuri, "A study of android application security." in *USENIX Security Symposium*, vol. 2, 2011, p. 2.

[21] W. Enck, P. Gilbert, S. Han, V. Tendulkar, B.-G. Chun, L. P. Cox, J. Jung, P. McDaniel, and A. N. Sheth, "Taintdroid: an information-flow tracking system for realtime privacy monitoring on smartphones," *ACM Transactions on Computer Systems (TOCS)*, vol. 32, no. 2, p. 5, 2014.

[22] H. Yin, D. Song, M. Egele, C. Kruegel, and E. Kirda, "Panorama: capturing system-wide information flow for malware detection and analysis," in *Proceedings of the 14th ACM conference on Computer and communications security (CCS)*. ACM, 2007, pp. 116–127.

[23] A. S. Uluagac, S. V. Radhakrishnan, C. Corbett, A. Baca, and R. Beyah, "A passive technique for fingerprinting wireless devices with wired-side observations," in *2013 IEEE Conference on Communications and Network Security (CNS)*. IEEE, 2013, pp. 305–313.

[24] C. Neumann, O. Heen, and S. Onno, "An empirical study of passive 802.11 device fingerprinting," in *2012 32nd International Conference on Distributed Computing Systems Workshops (ICDCSW)*. IEEE, 2012, pp. 593–602.

[25] L. H. Newman. (2018) Apple just made safari the good privacy browser. [Online]. Available: https://www.wired.com/story/apple-safari-privacy-wwdc

[26] T. Kohno, A. Broido, and K. C. Claffy, "Remote physical device fingerprinting," *IEEE Transactions on Dependable and Secure Computing*, vol. 2, no. 2, pp. 93–108, 2005.

[27] V. Brik, S. Banerjee, M. Gruteser, and S. Oh, "Wireless device identification with radiometric signatures," in *Proceedings of the 14th ACM International Conference on Mobile Computing and Networking*. ACM, 2008, pp. 116–127.

[28] H. Bojinov, Y. Michalevsky, G. Nakibly, and D. Boneh, "Mobile device identification via sensor fingerprinting," *Computing Research Repository (CoRR)*, 2014. [Online]. Available: https://arxiv.org/abs/1408.1416

[29] J. Fridrich, "Digital image forensics," *IEEE Signal Processing Magazine*, vol. 26, no. 2, 2009.

[30] D. Valsesia, G. Coluccia, T. Bianchi, and E. Magli, "Compressed fingerprint matching and camera identification via random projections," *IEEE Transactions on Information Forensics and Security*, vol. 10, no. 7, pp. 1472–1485, 2015.

[31] A. Das, N. Borisov, and M. Caesar, "Do you hear what i hear? fingerprinting smart devices through embedded acoustic components," in *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security (CCS)*. ACM, 2014, pp. 441–452.

[32] ——, "Tracking mobile web users through motion sensors: attacks and defenses." in *Proceedings of the 2016 Network and Distributed System Security (NDSS) Symposium*, 2016.

APPENDIX

A. Nominal gain estimation

Figure 6 shows the estimated gyroscope nominal gain from two iPhone X devices. Both devices were at rest on a desk during the data collection. To show the results more clearly, The figure only presents a small range of data. As shown in the figure, the difference between the estimated gyroscope nominal gain from these two devices is tiny. Therefore, we can use the estimated nominal gain from one device to estimate the gain matrix of another device. It is also clear from the figure that the degree of fluctuations around the nominal gain is different for these two devices, which indicates that their gain matrix is different and may be used as a fingerprint.

B. Gain Matrix Estimations of an iPhone 5

The estimations of the gyroscope gain matrix of an iPhone 5 is shown in Fig. 7. Here, values in each column are corresponding to the nine gain matrix values estimated from one batch of data (five batches in total), subtracting the nominal gain (unit: 2^{-16} dps).

C. Screenshots of the GYROID Proof of Concept App

Screenshots of the GYROID proof of concept app is shown in Fig. 8.

D. Distribution of the GYROID

Results are presented in Fig. 9, 10, and 11.

Fig. 6: Estimation of the gyroscope nominal gain from two iPhone X devices. The dash line marks the estimated nominal gain

$$\begin{bmatrix} -39.862363137944158 & -39.815935280377744 & -39.822619103792022 & -39.807082121390522 & -39.817704006855820 \\ 36.746838108661962 & 36.725833955351305 & 36.751195821364234 & 36.752406611083359 & 36.751266990517692 \\ 20.321937017717612 & 20.276563465382658 & 20.302206619012743 & 20.304280744909118 & 20.293839012434400 \\ -25.775903878528148 & -25.826990327452950 & -25.865651673449992 & -25.824470693110083 & -25.830236948855958 \\ 19.070418546381916 & 19.087074948988629 & 19.124401486623356 & 19.120464786337834 & 19.117788287737312 \\ -29.059409132023614 & -29.082230352356362 & -29.029881179984351 & -29.043929037666555 & -29.056049474283022 \\ 31.344396673396361 & 31.364984433906649 & 31.347831819184396 & 31.359581874031452 & 31.366435779636262 \\ -35.131464999438137 & -35.144073074775363 & -35.126381870264737 & -35.138736688905318 & -35.140984092442856 \\ -74.091971345986167 & -74.128105301813775 & -74.093537886235026 & -74.123911828207383 & -74.131598201240195 \end{bmatrix}$$

Fig. 7: Gain Matrix Estimations of an iPhone 5

(a) (b)

Fig. 8: Screenshots of the GyroID proof of concept app

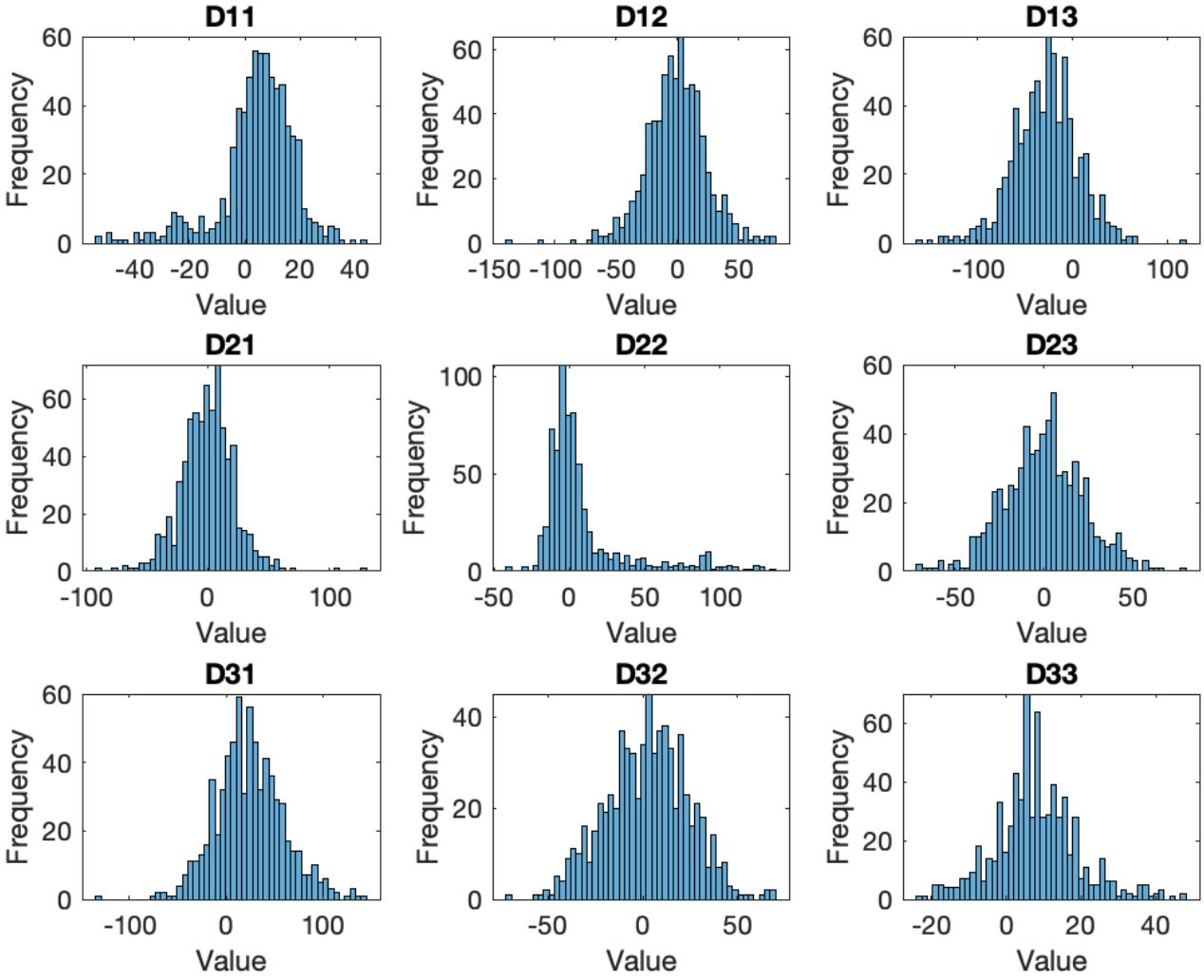


Fig. 9: Distribution of each element in the GYROID (nominal gain = 61 mdps)

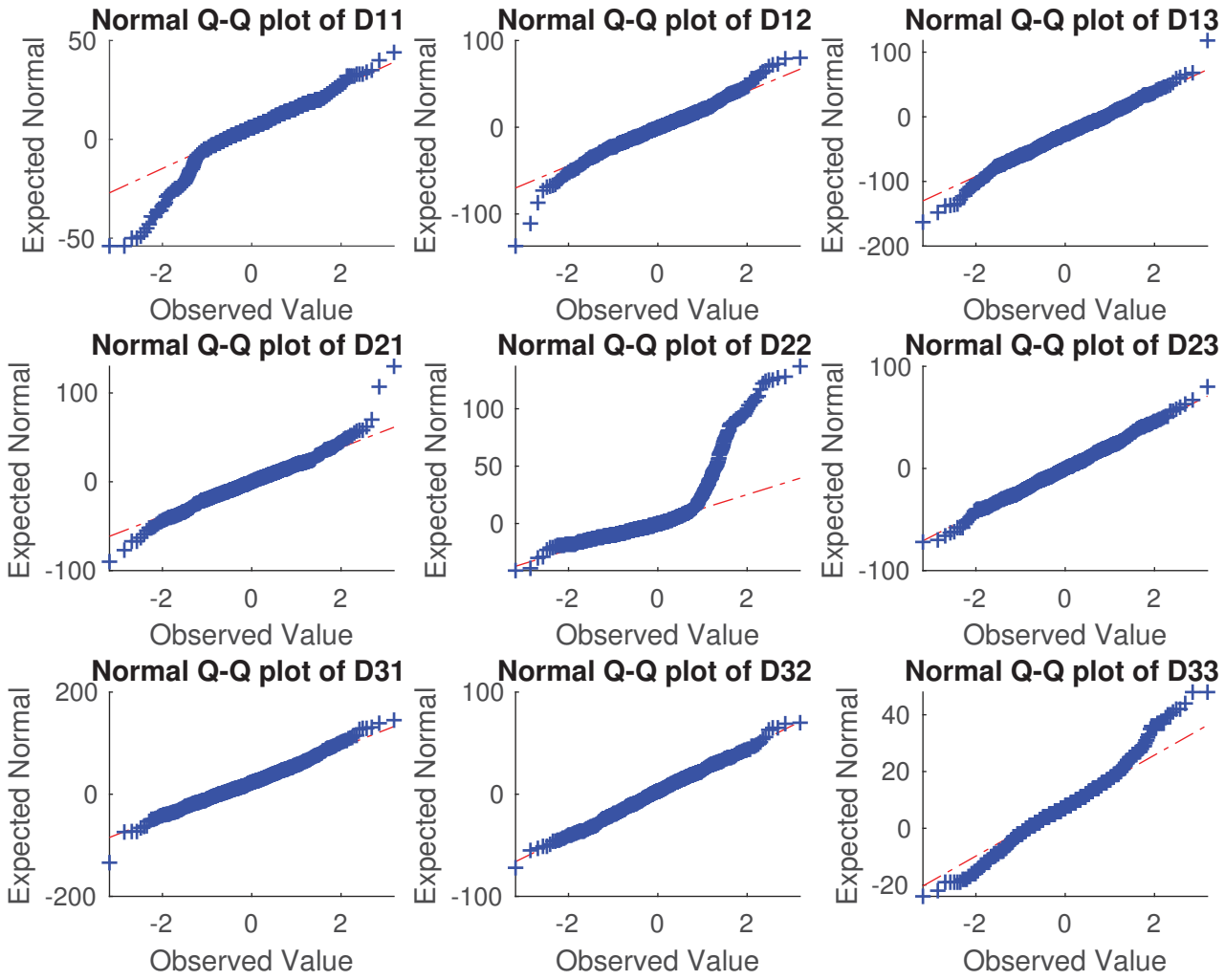


Fig. 10: Q-Q plot of each element in the GYROID (nominal gain = 61 mdps). If the distribution of a variable is normal, its Q-Q plot should appear linear

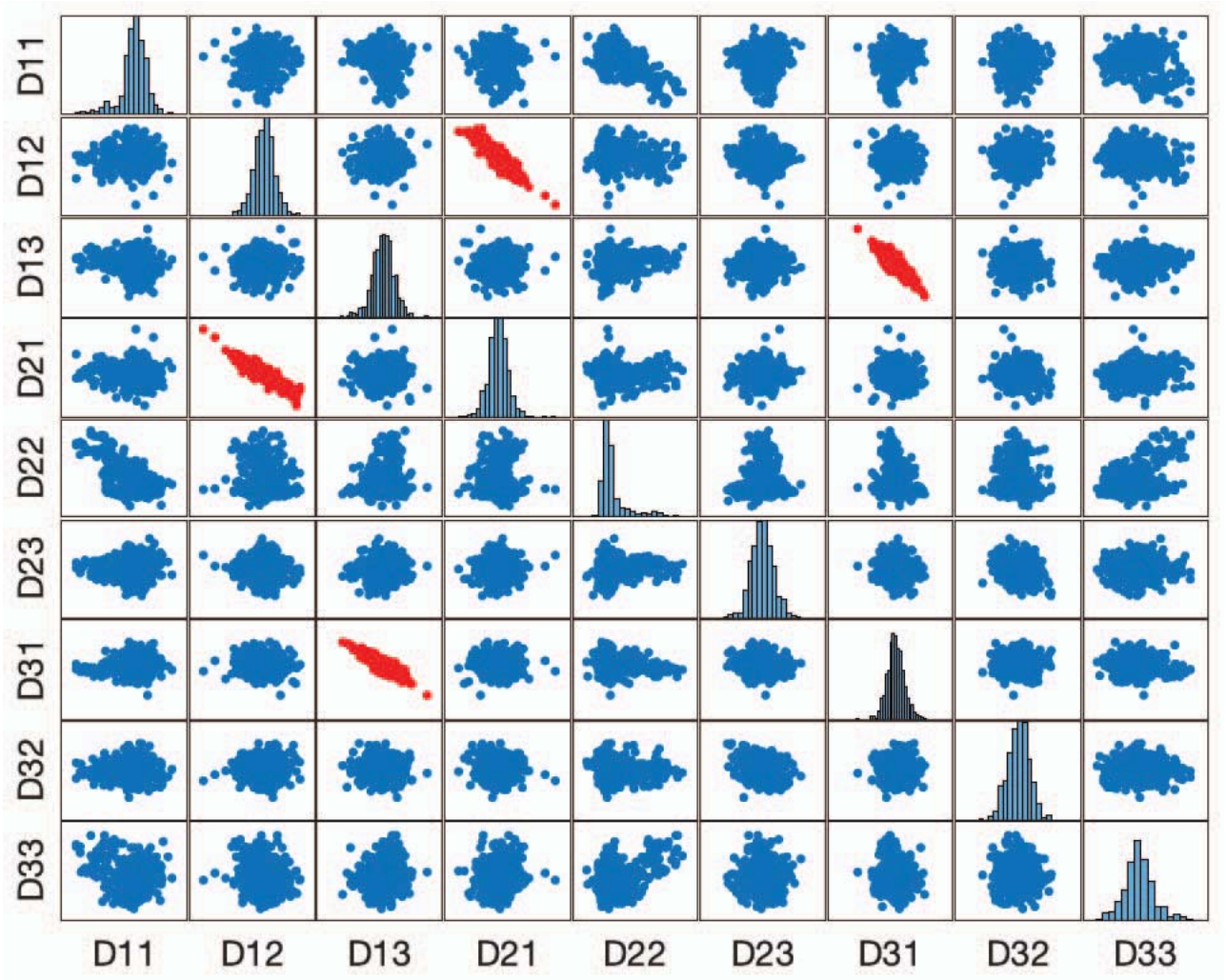


Fig. 11: Scatter plot matrix of elements in the GYROID (nominal gain = 61 mdps). Each scatter plot shows the relationship between two elements. Pairs with significant correlation are highlighted in red.