Factory Calibration Fingerprinting of Sensors

Jiexin Zhang, Alastair R. Beresford, and Ian Sheret

***Abstract*—*Device fingerprinting* aims to generate a distinctive signature, or *fingerprint*, that uniquely identifies individual computing devices. Fingerprints may be a privacy concern since apps and websites can use them to track user activity online. To protect user privacy, both Android and iOS have included a variety of measures to prevent such tracking. In this paper we present a new type of fingerprinting, *factory calibration fingerprinting*, that bypasses existing tracking protection. Our attack recovers embedded per-device factory calibration data from the accelerometer, gyroscope, and magnetometer sensors that are pervasive in modern smartphones by careful analysis of the sensor output alone. We discuss the factory calibration behaviour of each sensor and show that the calibration fingerprint is fast to generate, does not change over time or after a factory reset, and can be used to track users across apps and websites without any special permission from the user. We find the calibration fingerprint is very likely to be globally unique for iOS devices, with an estimated 67 bits of entropy for the iPhone 6S. In addition, we have analysed 146 Android device models from 11 vendors and found the attack also works on recent Google Pixel devices. For Pixel 4/4 XL, we estimate the calibration fingerprint provides about 57 bits of entropy. Following our disclosures, Apple deployed a mitigation in iOS 12.2 and Google in Android 11. We analyse Apple's fix and show that the mitigation is imperfect although it is likely to be sufficient in most threat models.**

***Index Terms*—Device fingerprint, factory calibration, motion sensor, mobile tracking, mobile privacy.**

I. Introduction

WHEN users visit a website, their web browser provides a range of information to the website, including the name and version of the browser, screen size, fonts installed, and so on. Ostensibly, this information allows the website to provide a great user experience. Unfortunately, this same information can also be used to track users. In particular, this information can be used to generate a distinctive signature, or device fingerprint, to identify users. Similarly, mobile app developers can also generate a device fingerprint using the idiosyncrasies of software and hardware.

Manuscript received June 22, 2020; revised September 15, 2020; accepted October 26, 2020. Date of publication November 20, 2020; date of current version December 21, 2020. This work was supported by Nokia Bell Labs. The work of Jiexin Zhang was supported by the China Scholarship Council. The work of Alastair R. Beresford was supported in part by the Engineering and Physical Sciences Research Council (EPSRC) under Grant EP/M020320/1. The associate editor coordinating the review of this manuscript and approving it for publication was Prof. Ulrich Rührmair. *(Corresponding author: Jiexin Zhang.)*

Jiexin Zhang and Alastair R. Beresford are with the Department of Computer Science and Technology, University of Cambridge, Cambridge CB3 0FD, U.K. (e-mail: jz448@cl.cam.ac.uk; arb33@cl.cam.ac.uk).

Ian Sheret is with Polymath Insight Ltd., Stevenage SG1 2DX, U.K. (e-mail: ian.sheret@polymathinsight.co.uk).

Digital Object Identifier 10.1109/TIFS.2020.3039685

Realising the potential risk to user privacy, both iOS and Android have included measures to prevent such tracking. On iOS, developers do not have access to the UDID (Unique Device IDentifier), IMEI (International Mobile Equipment Identity), and MAC address of hardware modules after iOS 7. Similar restrictions are also deployed on Android O; developers cannot get access non-resettable unique hardware identifiers even if users grant the app dangerous permissions such as the `READ_PHONE_STATE` permission. While it is still possible to track users by the advertising identifier on both iOS and Android, this method comes with several drawbacks. First, both platforms allow users to reset this identifier at any time and iOS also provides an option to limit access to this identifier. Moreover, apps requesting this identifier but do not serve any in-app advertisements will be rejected by the App Store. Last but not least, the advertising identifier is not accessible from mobile browsers. Similarly, although Android still allows apps to access the `ANDROID_ID`, it cannot be obtained from a website and its value is scoped by the signing key and user since Android 8. Different apps on a device will have different values of `ANDROID_ID` and a factory reset or an APK signing key change may also change its value [1]. Thus, it cannot be used to track users across apps and websites.

We have developed a new type of fingerprinting attack, the *factory calibration fingerprinting attack*, which can bypass these restrictions. Modern mobile devices are shipped with a variety of embedded motion sensors that apps rely on to provide rich functionality, including workout tracking, and improved user interaction. Natural variation during the manufacture of embedded sensors means that the output of each sensor is unique and therefore such natural variation may be exploited to create a device fingerprint.

Previous work applies machine learning techniques directly to sensor data in an attempt to create device fingerprints for smartphones; this has been shown to be ineffective (§VII). In our attack, instead of feeding sensor outputs into machine learning algorithms, we infer the per-device factory calibration data from the output of gyroscope, accelerometer, and magnetometer to construct a globally unique fingerprint. Overall, our attack has the following advantages:

Practical	the attack can be launched by any website or any app on a vulnerable device without requiring explicit confirmation or interaction by the user.
Efficient	the attack takes less than one second to generate a fingerprint.
Unique	the attack generates a globally unique fingerprint for vulnerable devices.

Robust	the calibration fingerprint never changes, even after a factory reset.
Effective	the attack provides an effective means to track users as they browse across the web and move between apps on their device.

In our previous research we have focused mostly on iOS devices and demonstrated its effectiveness on gyroscope and magnetometer data available in iOS [2]. This paper extends our previous work: we present our latest findings on accelerometer calibration on iOS devices (§III-E); we conduct a large-scale factory calibration behaviour analysis on popular Android device models (§III-F); we compare the calibration fingerprint with the Fingerprintjs2 fingerprint for Google Pixel phones (§IV-B); we analyse the calibration fingerprint for vulnerable Pixel devices and estimate entropy for each model (§V-B).

We followed a coordinated disclosure procedure and reported the vulnerability to Apple on 3rd August 2018 and Google on 10th December 2018. In iOS 12.2, Apple adopted our suggestion and added random noise to sensor outputs (CVE-2019-8541) while Google decided to round sensor outputs to a multiple of nominal gain in Android 11. In addition, Apple removed access to motion sensors from Mobile Safari by default and in later versions also removed motion sensor access from WebKit. However, in this paper we show that Apple's fix is imperfect since a calibration fingerprint can still be extracted if more sensor data is available (§VI).

We make the following contributions in this paper:

1) We introduce a new method of fingerprinting a device: the factory calibration fingerprinting attack.
2) We describe how factory calibration data can be extracted from the accelerometer, magnetometer, and gyroscope found on recent smartphones.
3) We demonstrate that the factory calibration data of the magnetometer and gyroscope together form a reliable fingerprint for iOS devices that does not change after factory reset or operating system updates.
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 calibration fingerprint 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 conduct a large-scale analysis on motion sensor calibration in 146 Android device models from 11 vendors. We find that all Google Pixel phones except for Pixel 1/1 XL can be fingerprinted by our attack; we show that the calibration fingerprint of the Pixel 4/4 XL has about 57 bits of entropy.
7) We analyse Apple's fix and show that it is imperfect: with $\sim$ 50K samples the exact fingerprint can be extracted.

II. 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, while the magnetometer measures the Earth's magnetic field relative to the device. These sensors are pervasive in modern mobile devices. 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 categorised 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 modelled stochastically; deterministic errors are produced by manufacturing imperfections and can be classified into three categories: *bias*, *scaling factor*, and *nonorthogonality misalignment errors* [3], [4].

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 [5]. In the context of mobile devices, the main benefit of per-device calibration is that it allows more accurate attitude estimation [6]. By contrast, sensors embedded in low-cost smartphones are usually poorly calibrated due to the high cost and complexity of factory calibration [7]. For an individual manufacturer, the choice of sensor calibration is, therefore, an engineering trade-off.

MEMS sensors usually convert and store the analogue measurement in a digital register through an Analogue-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 [8]:

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

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 [9]. The above equation can be further simplified as:

$$\mathbf{O} = \mathbf{G}(\mathbf{A} + \mathbf{B}) \tag{1}$$

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

A myriad of calibration techniques has been proposed to calculate the gain matrix and bias vector during manufacture [3]. Vendors can also 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

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.

time [10], [11]. 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 in iOS devices; the calibration fingerprinting for other sensors and devices is discussed later.

A. Gyroscope Calibration on iOS Devices

We have found that the gain matrix of the gyroscope in iOS devices is factory calibrated and further estimated the nominal gain for different device models based on the gyroscope outputs [2]. In particular, Table I lists the nominal gain (in mdps, millidegrees per second) of the gyroscope for all the iOS devices that we have measured. The estimated nominal gain of 61 mdps indicates that the sensor is likely configured to a measurement range of $\pm 2\,000$ dps and resolution of 16 bits ($4\,000/2^{16} \approx 0.061$). Furthermore, we have found that the fractional part of motion sensor outputs in iOS only has 16-bit resolution. 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} degrees per second (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.[1]

B. Fingerprinting From Mobile Apps

In general, manufacturing imperfections introduce idiosyncrasies across different sensors. If factory calibration is carried out on a per-device basis, then the calibration matrices may also be unique. Therefore, the gain matrix $\mathbf{G}$ may be used as a robust device fingerprint.

The general process to recover the device calibration fingerprint is illustrated in Fig. 1, which consists of six major steps: *Data Collection*, *Data Preprocessing*, *ADC Value Estimation*, *Gain Matrix Estimation*, *Validity Check*, and *Fingerprint Generation.*

1) 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 $\mathbf{O} = [\mathbf{O}_1, \mathbf{O}_2, \cdots, \mathbf{O}_N]$ 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.

2) 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:

$$\Delta\mathbf{O} = [\mathbf{O}_2 - \mathbf{O}_1, \mathbf{O}_3 - \mathbf{O}_2, \cdots, \mathbf{O}_N - \mathbf{O}_{N-1}]$$

Then, we select only a subset of the data, $\Delta\mathbf{O}^{\epsilon_i}$, where the absolute value of all its elements is lower than a multiplication of the nominal gain:

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

where ϵ_i is the threshold. We start ϵ_i from 1 and double its value for each iteration (i.e., $\epsilon_{i+1} = 2\epsilon_i$) until the subset covers the whole dataset (e.g., $\Delta\mathbf{O}^{\epsilon_i} = \Delta\mathbf{O}$). In each iteration, we feed $\Delta\mathbf{O}^{\epsilon_i}$ to the next step and update the value of $\widetilde{\mathbf{G}}$.

3) ADC Value Estimation: In this step, we aim to recover $\Delta\mathbf{A}^{\epsilon_i}$, which is the difference between consecutive ADC outputs. From Equation 1 we have:

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

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 estimate $\mathbf{G}$ by its ideal value $\mathbf{G}_0 = F_G \cdot \mathbf{I}$, where F_G is the nominal gain of the gyroscope. That is to say, we have $\widetilde{\mathbf{G}} = \mathbf{G}_0$ in the first iteration, where $\widetilde{\mathbf{G}}$ is the estimated gain matrix. Deterministic errors are comparatively small, and thus $\mathbf{G}$ should be relatively close to $\mathbf{G}_0$. Since $\Delta\mathbf{A}^{\epsilon_i}$ only has integer values, we can estimate $\Delta\mathbf{A}^{\epsilon_i}$ by:

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

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

$$\mathbf{E}^r = |\widetilde{\Delta\mathbf{A}^{\epsilon_i}} - \widetilde{\mathbf{G}}^{-1}\Delta\mathbf{O}^{\epsilon_i}|$$

To ensure the estimated values are correct, we require that every value in $\mathbf{E}^r_k$, which means the k-th column in $\mathbf{E}^r$, be lower than a threshold Γ (e.g., 0.1). If not, we believe the rounding is ambiguous and thus remove both $\widetilde{\Delta\mathbf{A}^{\epsilon_i}_k}$ and $\Delta\mathbf{O}^{\epsilon_i}_k$ from the dataset. Once all ambiguous values are removed, $\widetilde{\Delta\mathbf{A}^{\epsilon_i}}$ can be regarded as a safe estimate of $\Delta\mathbf{A}^{\epsilon_i}$. Nevertheless, if the device is moving rapidly (e.g., vigorous shaking),

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

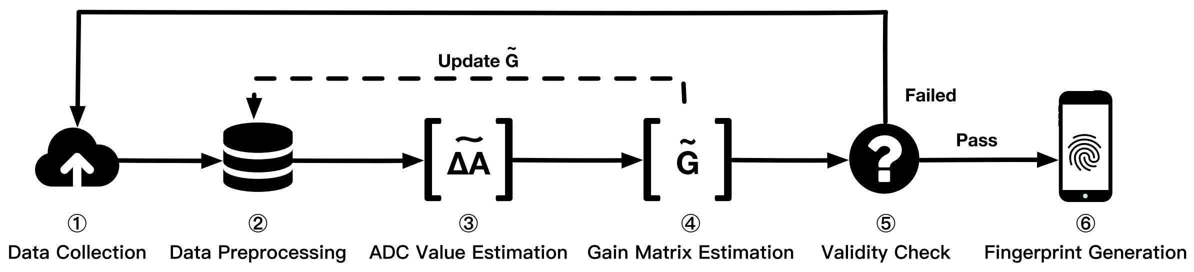


Fig. 1. General steps to recover the device calibration fingerprint.

rounding errors could be accumulated and cause rounding to an incorrect integer value (i.e., $\widetilde{\Delta \mathbf{A}^{\epsilon_i}} \neq \Delta \mathbf{A}^{\epsilon_i}$).

4) Gain Matrix Estimation: After estimating the ADC value matrix $\widetilde{\Delta \mathbf{A}^{\epsilon_i}}$, we can update the gain matrix estimate by:

$$\widetilde{\mathbf{G}} = \arg\min_{\mathbf{G}} \left\| \mathbf{G}\widetilde{\Delta \mathbf{A}^{\epsilon_i}} - \Delta \mathbf{O}^{\epsilon_i} \right\|_2^2$$

where $\|\cdot\|_2^2$ is the squared Euclidean 2-norm function. Or in words, we use the least squares solution to $\mathbf{G}\widetilde{\Delta \mathbf{A}^{\epsilon_i}} = \Delta \mathbf{O}^{\epsilon_i}$ as the gain matrix estimate.

After each update of $\widetilde{\mathbf{G}}$, we check if we have processed all the output data. If so, 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 range of data will be processed with $\epsilon_{i+1} = 2\epsilon_i$.

By way of an example, the $\widetilde{\mathbf{G}}$ that we estimated from an iPhone XS, in the units of 2^{-16} dps, is presented in Equation (2).

These numbers are extremely close to whole integers, indicating that the gain matrix is stored in the units of 2^{-16} dps. In fact, we found that all iOS devices with an M-series coprocessor store the gain matrix this way. For these devices, we can simply round $\widetilde{\mathbf{G}}$ to obtain the true gain matrix $\mathbf{G}$.

5) 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:

$$\mathbf{E}^e = \frac{\left\| \Delta \mathbf{O} - \widetilde{\mathbf{G}}\widetilde{\Delta \mathbf{A}} \right\|_2}{N-1}$$

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 $\mathbf{G}$. Otherwise, it means the ADC value estimation is incorrect, which is likely a result of vigorous movement during the data collection. In this case, we need to collect another batch of data and repeat the steps.

6) Fingerprint Generation: The generation of the gyroscope calibration fingerprint, GYROID, can be categorised 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:

$$\text{GYROID} = \text{round}(\widetilde{\mathbf{G}}) - \text{round}(\mathbf{G}_0) \tag{3}$$

where both $\widetilde{\mathbf{G}}$ and $\mathbf{G}_0$ are in the units of 2^{-16} dps. 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 the previous example is:

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

When the device does not contain an M-series coprocessor, the GYROID is calculated by:

$$\text{GYROID} = \widetilde{\mathbf{G}} - \text{round}(\mathbf{G}_0)$$

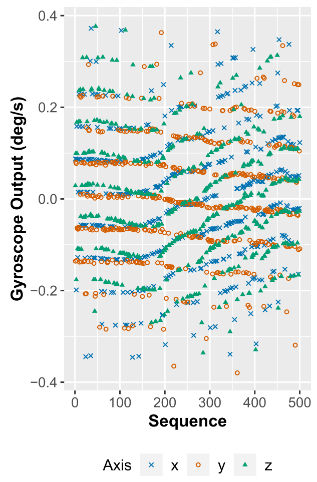
because values in the gain matrix are not simply integers in the units of 2^{-16} dps in this case.

7) Summary: In this section we present the general idea and procedure to generate a calibration fingerprint. Overall, the calculations are light-weight and are easy to implement. The attack works well under normal device interaction (e.g., device is resting on a desk or held in hand when browsing a webpage); it typically requires only 100 data samples that can be collected in less than 1 second to generate the GYROID (§IV). If the device is moving vigorously when collecting data, our approach will detect fast movement, as the *Validity Check* will fail, and keep trying until the movement is reduced. This attack is therefore practical since people rarely shake their device continuously and vigorously for extended periods.

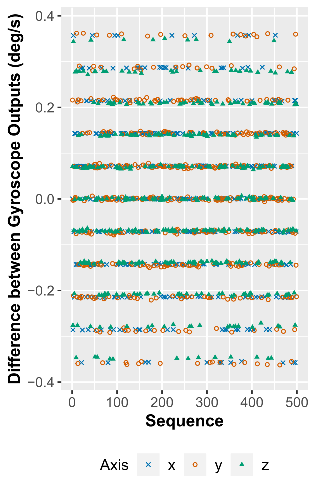
C. Fingerprinting From Mobile Websites

JavaScript also provides APIs for web developers to access the *fused* gyroscope data. In this paper, we state the sensor data is *fused* if its value is a result of applying a time-varying bias correction to the raw sensor data. In practice, it is common to use a Kalman filter to combine, or *fuse*, the accelerometer and gyroscope inputs to calculate the corrected bias values. Fig. 2 (a) presents the 500 sequential gyroscope samples collected from an iPad Air through mobile Safari when the device is at rest on a desk. As shown in Fig. 2 (a), quantisation 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 due to the bias correction.

$$\begin{bmatrix} 4012.000000000001 & -35.999999999999318 & -10.999999999999677 \\ 11.000000000000174 & 4030.999999999999 & 21.999999999999631 \\ -3.999999999999980 & -25.000000000000011 & 4016.000000000000 \end{bmatrix} \tag{2}$$



(a) Fused gyroscope data



(b) After differencing

Fig. 2. Gyroscope data collected via JavaScript (iPad Air).

Fig. 2 (b) shows that the bias part can be nearly eliminated by subtracting consecutive samples. Therefore, we can apply the same technique described in §III-B to recover the gain matrix.

D. 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 (e.g., 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.

Specifically, for apps running on a device with an M-series motion coprocessor, they can follow the steps in §III-B to recover the exact GYROID. Otherwise, if the device does not have a motion coprocessor (e.g., iPhone 4/4S/5) or adversaries only have access to fused gyroscope data (e.g., via JavaScript), there are two options to determine whether two GYROID entries in the database represent the same physical device.

- Option 1 (Clustering): adversaries 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, they 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 adversaries know they came from the same device (the entropy of $\mathbf{G}$ is discussed in §V-A).
- Option 2 (Rounding): adversaries can still use Equation 3 to generate the GYROID. However, the estimated GYROID may deviate from the true one by at most ± 1 for each of the 9 values. Therefore, adversaries can store the estimated GYROID 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.

E. Fingerprinting Other Motion Sensors

1) Magnetometer: We found that the magnetometer in iOS devices can also be fingerprinted. Similar to the gyroscope, the raw readings from the magnetometer only have a resolution of 2^{-16} μT (microtesla). 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 slightly different sensitivity for each axis.
- Type II (fixed sensitivity, moderate fluctuation): Type II devices have the same sensitivity for every axis but there is a moderate fluctuation within each cluster.
- Type III (different sensitivity, moderate fluctuation): Type III devices have a slightly different sensitivity for each axis and there is a moderate fluctuation within each cluster; the quantisation of the data is 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 quantisation of the data is not as evident as in other cases.

TABLE II
MAGNETOMETER TYPE OF DIFFERENT IOS DEVICE MODELS

Type	Model	Nominal Gain (μT)
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.

We summarise the magnetometer type of different iOS 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-B to obtain the magnetometer fingerprint (i.e., MAGID). Although the gain matrix of the magnetometer is not stored at 2^{-16} μT resolution, adversaries can use the same techniques discussed in §III-D 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 as a finer-grained fingerprint when analysing apps.

2) Accelerometer: We observed a similar quantisation pattern for the accelerometer in older generations of iOS devices, including the iPhone 4S and iPad Mini. For example, Fig. 3 shows the consecutive differences between 2 000 accelerometer outputs from an iPhone 4S. The quantisation is clear and thus we can apply the same approach described in §III-B to recover the accelerometer fingerprint. Note that these devices do not use an Apple motion coprocessor, and thus adversaries would need to use techniques described in §III-D to launch the attack. For newer versions of iOS devices, such as the iPhone 5 also shown in Fig. 3, Apple uses a higher-resolution accelerometer that conceals the quantisation in outputs. Our attack currently does not directly apply to these devices.

In this paper, we define the SENSORID as a combination of distinctive sensor calibration fingerprints. In the case of

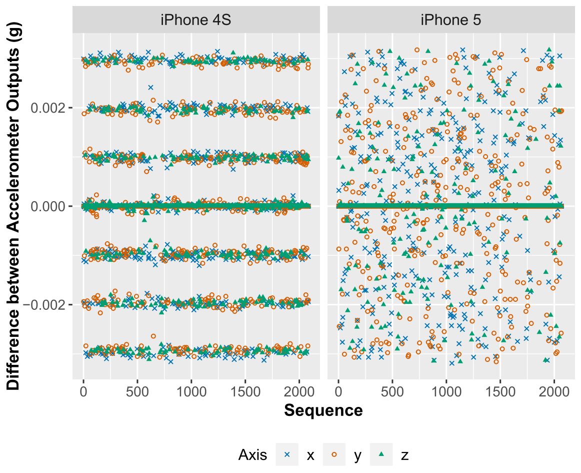


Fig. 3. Comparison between iPhone 4 and 5 (Accelerometer).

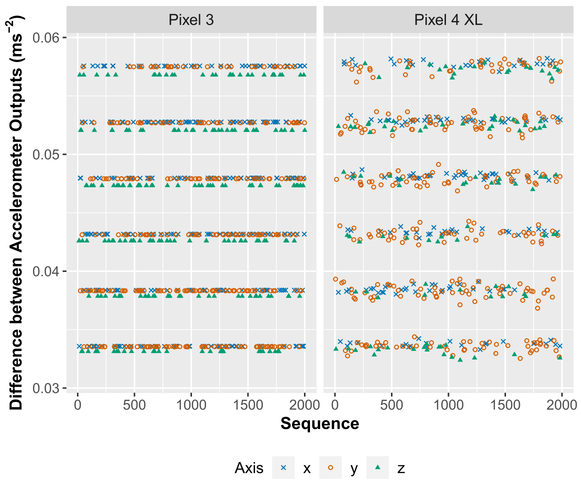


Fig. 4. Comparison between Pixel 3 and 4 XL (Accelerometer).

iOS devices, the SENSORID includes both the GYROID and MAGID. For older versions of iOS devices (e.g., iPhone 4S and iPad Mini), the SENSORID also includes the accelerometer fingerprint (i.e., ACCID).

F. Fingerprinting Android Devices

To study whether Android devices are susceptible to similar fingerprinting attacks, we used four automated testing platforms to collect data from 146 Android device models from 11 vendors: BQ, Google, HTC, Huawei, LG, Motorola, Nokia, OnePlus, Samsung, Sony, and Xiaomi (§IV-B). Among all the Android devices we have tested, we found that all Google Pixel phones, other than the Pixel 1/1 XL, can be fingerprinted by our approach; we did not observe per-device calibration behaviour on the Pixel C tablet. In addition, we noticed that the calibration process applied to motion sensors varies across device models. In particular, we found the full gain matrix of both the accelerometer and gyroscope in Google Pixel 4 and 4 XL are per-device calibrated, while only the leading diagonal of the gain matrix for the accelerometer is calibrated in other Pixel devices.

Fig. 4 shows the consecutive difference between 2 000 accelerometer outputs collected from a Pixel 3 and a Pixel 4 XL when they were at rest on a desk. The quantisation in the accelerometer outputs is clear in both cases. In addition, the figure suggests that only the scale matrix of the accelerometer is calibrated in Pixel 3 (i.e., only main-diagonal elements in the gain matrix are calibrated); the same pattern is also observed in Pixel 2/2 XL/3 XL/3a/3a XL. Fig. 4 also suggests that all 9 values in the gain matrix are calibrated in the Pixel 4 XL; the same pattern is also observed in the Pixel 4. Apart from the accelerometer, the gyroscope in Google Pixel 4 and 4 XL is also per-device calibrated.

TABLE III
SENSORID COMPOSITION

SENSORID	Device Model
ACCID	iPhone 4S, iPad Mini Pixel 2/2 XL/3/3 XL/3a/3a XL/4/4 XL
GYROID	iOS devices, Pixel 4/4 XL
MAGID	iOS devices

In addition to Pixel devices, we have noticed that the accelerometer in Huawei Honor 20 Lite, MediaPad M3 Lite 10, and MediaPad T3 10 and the gyroscope in BQ Aquaris X2 have different sensitivity in each axis. However, we only have one device for each of these models and thus cannot confirm whether they are per-device calibrated. So far we have tested 321 unique devices (146 unique device models); we did not observe per-device calibration behaviour on any other Android device models we tested. We therefore focus on Google Pixel devices in the rest of the paper. The choice of factory calibration is up to individual manufacturers.

For vulnerable Google Pixel phones, we can extract the SENSORID from both an Android app and a website running on an Android browser. In the case of Google Pixel 4 and 4 XL, the SENSORID includes both the ACCID and GYROID. For other Pixel models excluding Pixel 1, the SENSORID only includes the ACCID. In summary, Table III lists all device models whose SENSORID includes the ACCID, GYROID, and MAGID, respectively.

IV. EVALUATION

A. Fingerprinting iOS Devices

We developed both a website and an iOS app to collect sensor data. The iOS app collects raw data from the motion sensors (accelerometer, gyroscope, and magnetometer) at 200 Hz and does not ask users to put the device in any particular position. The app also embeds a WebView; the embedded WebView and the separate website both collect fused accelerometer and gyroscope data via JavaScript.

For both the app and the website, we use the Fingerprintjs2 [12] library in the default configuration to generate a browser fingerprint for evaluation purposes. In addition to volunteers, we recruited participants from Amazon Mechanical Turk[2] and Prolific[3] 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 at the University of Cambridge.

[2]https://www.mturk.com
[3]https://prolific.ac

TABLE IV
COMPARISON OF iOS DEVICE FINGERPRINTS

# Devices	Fingerprint	Group Size	# Groups
870	GYROID	1	870
	Fingerprintjs2	1	391
		2–36	96
		45	1
795	MAGID	1	775
		2	10
	Fingerprintjs2	1	308
		2–36	97
		45	1
10	ACCID	1	10
	Fingerprintjs2	1	8
		2	1

To date, the SENSORID app has collected data from 795 unique iOS devices; 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. On both the app and the website, we ask users to tell us whether they have submitted the data from this device before.

Using the raw gyroscope data collected from the 761 iOS devices with an M-series coprocessor, we are able to recover the exact GYROID. For the other 34 devices that do not contain an M-series coprocessor, we use the rounding option in §III-D 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-D 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 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 analyse 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 IV. 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. We find that the 10 groups of size 2 are all old device models with a Type I magnetometer, indicating they 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.

Similar to the analysis of data collected from the built-in WebView, we use the clustering option in §III-D to analyse the accelerometer fingerprint and apply MANOVA to identify unique devices. As discussed in §III-E, our attack only applies to the older generations of iOS devices. In our dataset, that includes 9 iPhone 4S devices and an iPad 3. We apply our attack on these 10 devices and find that all of them have a unique ACCID. It is likely that other iOS devices prior to iPhone 4S can also be attacked by our approach, but we do not have data from these devices to confirm it.

Finally, we compare the GYROID, MAGID, ACCID with the default configuration of Fingerprintjs2, which utilises font detection, canvas, WebGL, etc to fingerprint devices. Table IV presents the results and demonstrates GYROID, MAGID, and ACCID 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; these 45 devices are all from the UK. In the case of ACCID, it identifies all 10 unique devices, while Fingerprintjs2 generates the same fingerprint for two iPhone 4S devices; both devices are from Germany. The results indicate that the Fingerprintjs2 fingerprint may be correlated with a particular handset configuration.

We have also developed a proof of concept app for iOS devices with an M-series motion coprocessor. The app implements our attack to generate the GYROID of the test device. The code is written in Swift 4.1 with XCode 9.4.1. 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. Overall, it takes about 0.5 seconds to collect 100 gyroscope samples and another 0.01 seconds to generate the GYROID. Vigorous movement during extraction may require additional samples, but the task nevertheless completes within a few hundred samples and takes a few seconds. In any case, the generated GYROID always stays the same. A proof-of-concept webpage and demo videos can be found on our website: https://sensorid.cl.cam.ac.uk.

B. Fingerprinting Google Pixel Devices

In general, it is difficult to find many people with a Pixel device via crowdsourcing platforms due to the relatively small market share. Nevertheless, we found most online app testing platforms provide access to Pixel devices. Therefore, we developed an Android app to collect raw motion sensor data and send it back to our server. In a method similar to the one used in our iOS app, our app also embeds a WebView that collect web sensor data via JavaScript as well as a fingerprint generated by Fingerprintjs2. In addition, the app records the `ANDROID_ID` that is unique to each combination of the app-signing key, user, and device to identify unique devices in our dataset.

TABLE V
COMPARISON OF PIXEL DEVICE FINGERPRINTS

Device Model	Fingerprint	Group Size	# Groups
Pixel 2/2 XL	AccID	1	46
	Fingerprintjs2	1	17
		2	4
		4	1
		5	2
		7	1
Pixel 3/3 XL & Pixel 3a/3a XL	AccID	1	61
	Fingerprintjs2	1	25
		2	8
		3	5
		5	1
Pixel 4/4 XL	AccID	1	45
	GyroID	1	45
	Fingerprintjs2	1	10
		2	1
		9	1
		10	1
		14	1

During our experiment, we deployed the Android app on AWS Device Farm,[4] Firebase Test Lab,[5] App Centre,[6] and Sauce Labs[7] to collect data from various Android device models. Since we only found per-device calibration on Pixel phones other than the Pixel 1/1 XL, we focus our analysis on these devices. By the end, we have collected data from 152 unique Pixel devices.

Using the raw accelerometer data, we generate the AccID for each device and use clustering to determine whether two devices have the same AccID. We compare the results with Fingerprintjs2 in Table V. For Pixel 4 and 4 XL devices, we also calculate their GyroID and compared it with other fingerprints. As shown in the table, both AccID and GyroID uniquely identify every Pixel device while multiple devices have the same fingerprint generated by Fingerprintjs2. In particular, 14 out of 45 Pixel 4/4 XL devices have the same Fingerprintjs2 fingerprint. This is likely because the embedded WebView with default configuration does not expose many distinctive characteristics when two devices are running the same Android version. The fingerprint is likely to have more entropy if Fingerprintjs2 is running in an Android browser. Nevertheless, unlike SensorID, which is a hardware identifier, Fingerprintjs2 cannot track users as they move across Android browsers.

In addition, we have tried to generate the SensorID for each device using the web sensor data collected via JavaScript in the embedded WebView. We then differentiate devices based on their SensorID, generated from the web data, using clustering. We successfully identified all unique Pixel devices by their AccID. For Pixel 4 and 4 XL phones, we were also able to identify every individual device by their GyroID.

V. Discussion

In this section, we discuss some possible concerns regarding the validity of this research.

[4]https://aws.amazon.com/device-farm
[5]https://firebase.google.com/docs/test-lab
[6]https://appcentre.ms
[7]https://saucelabs.com

A. Is SensorID Unique for iOS Devices?

To study how unique is SensorID, we first study 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 choose 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 it 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. For simplicity, we denote the GyroID as $\mathbf{D} \in \mathbb{Z}^{3\times 3}$ in the following analysis.

1) Normality Analysis: 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 the non-diagonal elements in $\mathbf{D}$ have strong normality, while 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. The result suggests that we may need a finer-grained analysis for the main-diagonal elements. When we run a normality test on data from each device model separately we find that the main-diagonal elements also show strong normality.

2) Correlation Analysis: To test for correlation, we run the Pearson correlation test on each $\mathbf{D}_{ij}$ and find that $\mathbf{D}_{12}$ and $\mathbf{D}_{13}$ are strongly correlated with $\mathbf{D}_{21}$ and $\mathbf{D}_{31}$ at the 0.01 significance level, respectively. Therefore, we exclude $\mathbf{D}_{21}$ and $\mathbf{D}_{31}$ from our entropy calculation to avoid over-estimation.

3) 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, it is a result of rounding, and thus we can still use the 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:

$$\mathrm{H}(X) = -\sum_{x_i \in X} \mathrm{P}(x_i) \log_2 \mathrm{P}(x_i) \tag{5}$$

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 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:

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

By this equation, we calculate the entropy of $\mathbf{D}_{12}$, $\mathbf{D}_{13}$, $\mathbf{D}_{23}$, and $\mathbf{D}_{32}$. For main-diagonal elements (i.e., $\mathbf{D}_{11}$, $\mathbf{D}_{22}$, and $\mathbf{D}_{33}$), we 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). For these iPhone 6S devices, we adopt a similar approach and apply Equation 5 to calculate the entropy. As a result, we estimate the GyroID for iPhone 6S has about 42 bits of entropy.

By the same analysis, we estimate the entropy of the MagID for iPhone 6S. If adversaries launch the attack using the rounding option (§III-D), 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. Since we only have 10 old-generation iOS devices that have an AccID, we do not include AccID into the SensorID entropy calculation. We observe no evidence of strong correlation between the MagID and GyroID. Therefore, we estimate the SensorID for iPhone 6S has around 67 bits of entropy.

4) Uniqueness Analysis: There were 728M active iPhones worldwide in April 2017 and the iPhone 6S devices accounted for 18% of them [13]. 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.

5) Limitations: Results from both the Kolmogorov-Smirnov and Shapiro-Wilk normality tests suggest that values in $\mathbf{D}$ are consistent with a normal distribution, but it is possible that the actual distribution is not normally distributed in the tail regions. For example, manufacturers may discard sensors that have an extreme value in the gain matrix; this would reduce the available entropy. We therefore need to consider whether non-normal distributions might invalidate our entropy calculations.

Firstly, it is worth mentioning that this kind of rejection policy is unlikely in practice; one of the key benefits of factory calibration is that sensors with anomalous physical gains will still perform well when calibrated. More importantly, the calculation of entropy is dominated by the core shape of the distribution, where we have abundant data. Non-Gaussianity may affect the tails of the distribution, but this would have a negligible effect on the calculated entropy. To give a concrete example: we found all values in $\mathbf{D}$ fall inside the range $(\mu - 4\sigma, \mu + 4\sigma)$. If we make the assumption that values outside this range are discarded, we still estimate the SensorID provides around 67 bits of entropy for the iPhone 6S.

A related concern is there could be undetected higher-order correlations between values in $\mathbf{D}$. A similar argument applies in this case: the entropy calculation is dominated by the core of the (now multivariate) distribution, where we have abundant data, and where we see no evidence of non-independence. In the tail regions, non-independence might go undetected, but this would have little impact on the calculated entropy.

Ultimately, the calculation of entropy cannot be done with absolute rigour given a finite number of samples from an unknown distribution, but it is still possible to perform a thorough analysis, and significant errors in the estimated entropy of SensorID due to non-Gaussianity or non-independence are very unlikely.

B. Factory Calibration in Android Devices

Rooted Android handsets provide access to the gain matrix values for the motion sensors in the local file system on boot. We therefore confirmed that our distinct estimates for the AccID on two Pixel 3 devices were correct as well as our estimated AccID and GyroID values for a Pixel 4 device. While the magnetometer in the Pixel 3/4 also has a full gain matrix, its values appear to be the same for all devices of the same model and thus does not provide any entropy. The motion sensors in other Android devices may also be factory calibrated. If the calibration is restricted to offsets (i.e., bias compensation) then our approach is ineffective since it targets the gain matrix and cannot recover bias compensation.

To estimate the entropy of SensorID for vulnerable Pixel devices, we group these devices into three categories: Pixel 2 series (Pixel 2/2 XL), Pixel 3 series (Pixel 3/3 XL/3a/3a XL), and Pixel 4 series (Pixel 4/4 XL). For each category, we analyse the normality of each value and the correlation between values in the fingerprint. For Pixel 2 and 3 series, the AccID only has non-zero values in the leading diagonal ($\mathbf{D}_{11}$, $\mathbf{D}_{22}$, $\mathbf{D}_{33}$), and thus it provides less entropy than the Pixel 4 series. For Pixel 4 series, we find that some off-diagonal values in the AccID/GyroID are strongly correlated with each other. Thus, we only keep one of the dependent variables in the entropy calculation to avoid over-estimation.

Using a similar method to the one used in our previous work on the iOS magnetometer, we estimate the entropy based on the assumption that adversaries choose to attack using the rounding option, which has ± 1 uncertainty for each element. Results are presented in Table VI. In addition, we find off-diagonal elements in GyroID are strongly correlated with off-diagonal elements in AccID in Pixel 4 series devices. This is likely because the accelerometer and gyroscope are integrated into the same chip (`LSM6DSR`). Therefore, to estimate the entropy of SensorID, the combination of AccID and GyroID, we simply add the entropy provided by the main-diagonal variables in the AccID to the entropy provided by the GyroID. As a result, we estimate the SensorID provides around 57 bits of entropy; a precise estimate of entropy is difficult since we only have data from 45 Pixel 4/4 XL devices whereas in our previous study with iOS we used data from 127 iPhone 6S devices. Assuming that our entropy estimate is accurate, and analysing this as an example of the birthday problem, if there are 100 million Pixel 4/4 XL devices in the market, then the probability that every device has a globally unique SensorID is around 97%.

C. 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

TABLE VI
SENSORID ENTROPY ESTIMATION (PIXEL DEVICES)

Fingerprint	Device Model	# Devices	Entropy (bits)
ACCID	Pixel 2/2 XL	46	~14
	Pixel 3/3 XL/3a/3a XL	61	~12
	Pixel 4/4 XL	45	~25
GYROID	Pixel 4/4 XL	45	~45

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.

D. Consistency of SENSORID

We have not observed any change in the SENSORID of our test devices in the past 16 months. Our dataset includes iOS devices running iOS 9/10/11/12 and Pixel devices running Android 8/9/10. We have tested compass calibration, factory reset, and updating operating system; 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.

E. Impact and Responsible Disclosure

We followed a responsible disclosure procedure and reported this vulnerability to Apple on 3rd August 2018 and Google on 10th December 2018. In particular, we suggested two possible countermeasures. The first is to add a random noise $\epsilon \in \mathbb{R}^{3\times 1}$, from the uniform distribution in the range $[-0.5, 0.5]$, to each ADC output. The added noise obfuscates the effect of quantisation making the attack much harder. The second approach we proposed is to round the calibrated sensor output to the nearest multiple of the nominal gain. This approach is more practical to apply since it does not require access to the ADC values.

In iOS 12.2, Apple adopted our suggestion and added random noise to sensor outputs (CVE-2019-8541). In addition, Apple removed access to motion sensors from Mobile Safari by default and in later versions removed motion sensor access from WebKit as well. Recently, Google pushed a fix in Android 11 that rounds the motion sensor outputs to the nearest multiple of the nominal gain.

When running an iOS version prior to iOS 12.2, all iOS devices that have motion sensors can be fingerprinted by this approach, including the 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) were vulnerable to this calibration-based fingerprinting attack, even with the fingerprinting protection mode turned on. For Google Pixel phones running Android 10, we notice that some privacy-enhanced browsers, including Brave and Tor Browser, do block access to motion sensors by default while others (Chrome, Firefox, Firefox Focus, Opera, and Duckduckgo) do not. Using a browser that blocks motion sensor access could protect Pixel phone users from this attack when browsing online. 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 [14]. This is troublesome since it is likely that the SENSORID can be calculated with exfiltrated data, allowing retrospective device fingerprinting. The latest iOS devices which always run iOS 12.2 or later so the attack described does not work. Nevertheless, a dedicated attacker may still be able to extract the fingerprint (§VI).

VI. APPLE'S FIX

Apple declined to share the details of the fix deployed in iOS 12.2. Therefore we reverse-engineer Apple's fix by studying the gyroscope outputs in iPhone devices with iOS 12.2 or later; we show that the random noise applied does not fully conceal the calibration fingerprint.

A. Analysis of Apple's Fix

Fig. 5 presents the histogram of the raw gyroscope data in the y axis collected from the same iPhone X with two different iOS versions. In both cases, the device is at rest on a desk. When the device is running iOS 11.4.1, the quantisation in the histogram is clear and we can recover the exact gain matrix (§III-B). To mitigate our attack, Apple added random noise to the gyroscope outputs in iOS 12.2, concealing the quantisation information. Fig. 5 suggests that the added noise follows a uniform distribution, which is one of the countermeasures we proposed to Apple. However, we notice there are some peculiarities in the noise added.

To figure out the range of the uniform noise applied, we first obtain the gyroscope gain matrix of an iOS device when it is running an iOS version before iOS 12.2. Then, we update the device to the latest iOS version and take 20K gyroscope measurements from the device at 200 Hz when the device is at rest on a desk. We denote the gain matrix as $\mathbf{G}$ and these gyroscope outputs as $\mathbf{O}$. Then, the underlying bias-corrected ADC outputs $\mathbf{I}$ (i.e., $\mathbf{I} = \mathbf{A} + \mathbf{B}$) can be estimated by:

$$\widetilde{\mathbf{I}} = \text{round}(\mathbf{G}^{-1}\mathbf{O}) \tag{6}$$

Although the estimated ADC values may not be accurate due to perturbation, it gives us useful insights about the added noise. Furthermore, we can get the noise estimate, $\widetilde{\mathbf{N}}$, by:

$$\widetilde{\mathbf{N}} = \mathbf{O} - \mathbf{G}\widetilde{\mathbf{I}}$$

By way of an example, Fig. 6 presents the histogram of the estimated gyroscope noise of an iPhone XS. As shown in the Fig. 6, the majority of the estimated noise are distributed uniformly in the range $[-1997, 1997] \times 2^{-16}$ dps. The small number of outliers are likely produced due to the inaccurate estimation of the ADC values in Equation 6.

We have also tested other iOS devices and observed the same result. This confirms that Apple did not add random noise in the range $[-0.5, 0.5]$ to the ADC values as we proposed but instead added random noise in the range $[-1997, 1997] \times 2^{-16}$ dps to the calibrated signal. Because the width of the random noise is slightly narrower than the sensitivity of some gyroscope axes, it leaks more information

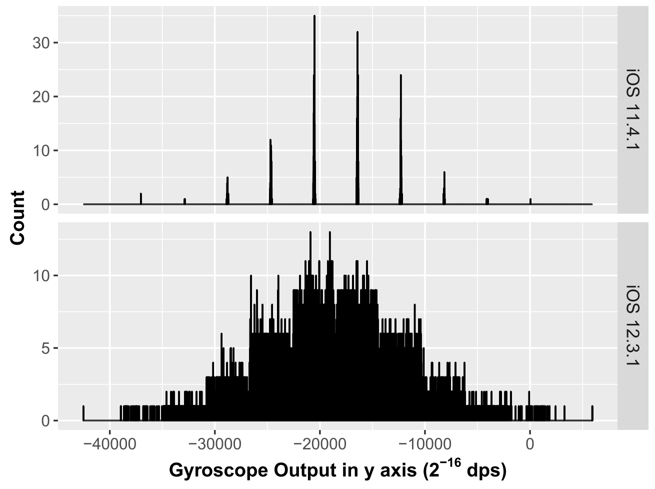


Fig. 5. Histogram of raw gyroscope data of an iPhone X.

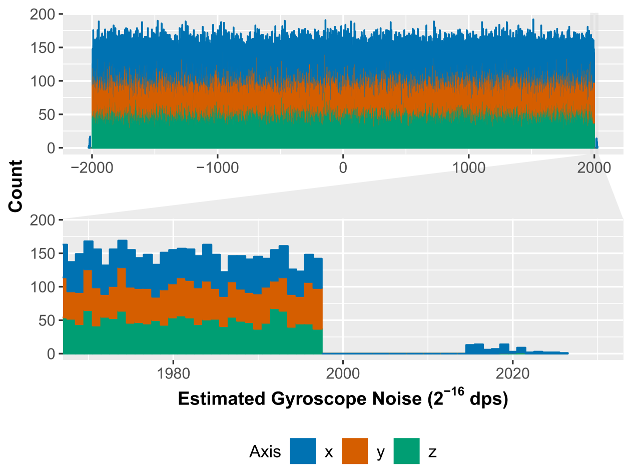


Fig. 6. Histogram of estimated gyroscope noises (iPhone XS).

than our original proposal. In the case of the iPhone XS in Fig. 6, the width of the perturbation, 3995, is lower than the sensitivity of all three axes (4012, 4031, and 4016, respectively). Here, we show that we can recover the gain matrix from the noisy data by performing a maximum likelihood search using simulated annealing.

B. Attack of Apple's Fix

In this section, we first define the objective function to quantify the likelihood of observing the outputs given a gain matrix. Then, we show that the exact gain matrix can be estimated from the objective function using simulated annealing.

1) Objective Function: For a candidate gain matrix $\widetilde{\mathbf{G}}$, we estimate the corresponding bias-corrected ADC outputs $\mathbf{I}$ by:

$$\widetilde{\mathbf{I}} = \text{round}(\widetilde{\mathbf{G}}^{-1}\mathbf{O})$$

Here, we did not subtract sensor outputs to remove the bias as we did in §III-B because it would spread the noise (double the noise range) and make the ADC value estimation less accurate. Both the $\mathbf{O}$ and $\widetilde{\mathbf{G}}$ are in the units of 2^{-16} dps so they only contain integer values. Nevertheless, the estimated $\widetilde{\mathbf{I}}$ is not guaranteed to be correct due to the perturbation, which is why we observed a few outliers in Fig. 6.

Then, we estimate the clean gyroscope outputs (i.e., without added noise) by:

$$\widetilde{\mathbf{O}} = \widetilde{\mathbf{G}}\widetilde{\mathbf{I}}$$

And the offset between the estimated outputs $\widetilde{\mathbf{O}}$ and observed outputs $\mathbf{O}$ can be calculated by:

$$\mathbf{\Delta} = \mathbf{O} - \widetilde{\mathbf{O}}$$

Since the added noise is uniformly distributed in the range $[-1997, 1997] \times 2^{-16}$ dps, any offset beyond this range is caused by either an incorrect gain matrix (i.e., $\widetilde{\mathbf{G}} \neq \mathbf{G}$) or incorrect ADC estimations (i.e., $\widetilde{\mathbf{I}} \neq \mathbf{I}$). To quantify the error in each data sample, we define the following error function:

$$f(\mathbf{\Delta}_i) = \sum_{\delta \in \mathbf{\Delta}_i} \max(|\delta| - 1997, 0)$$

Furthermore, we define the range-related likelihood function for each data sample as follows:

$$l_r(\mathbf{\Delta}_i) = \exp(-\frac{f(\mathbf{\Delta}_i)}{T})$$

where $\exp(\cdot)$ is the natural exponential function and T is the temperature variable used in simulated annealing that decreases over iterations.

Although the likelihood function $l_r(\mathbf{\Delta}_i)$ penalises data samples with an offset beyond the added noise range, it does not give us any information about the distribution of data within the range. In general, the ADC outputs of the gyroscope in every axis follow a normal distribution when the device is resting on a platform such as a desk (as shown in Fig. 5). If the device has moved during the data collection, we can use a stationary position filter to get the segments with stationary measurements. Therefore, we can fit a normal distribution $N(\mu_a, \sigma_a)$ to $\widetilde{\mathbf{I}}_a$ for each axis $a \in \{x, y, z\}$. Then, based on the normal distribution, we can calculate the distribution-related likelihood function, $l_d(\mathbf{I}_i)$, by:

$$l_d(\mathbf{I}_i) = \prod_{a \in \{x,y,z\}} p_a(I_{i_a})$$

where $p_a(\cdot)$ is the probability density function of the normal distribution $N(\mu_a, \sigma_a)$ and I_{i_a} is the output $\mathbf{I}_i$ in the axis a.

Finally, we define our objective function $L(\mathbf{O}|\widetilde{\mathbf{G}})$ as a negative log likelihood function:

$$L(\mathbf{O}|\widetilde{\mathbf{G}}) = -\sum_{i=1}^{N} \log(l_r(\mathbf{\Delta}_i) l_d(\mathbf{I}_i))$$

where N is the number of gyroscope outputs. Then, the true gain matrix can be estimated by the following equation:

$$\mathbf{G} = \arg\min_{\widetilde{\mathbf{G}}} L(\mathbf{O}|\widetilde{\mathbf{G}}) \tag{7}$$

2) Simulated Annealing: Simulated annealing is a probabilistic technique for solving optimisation problems and is often used in the presence of large numbers of local optima [15]. Since the objective function in Equation 7 is highly non-smooth, we use simulated annealing to solve this optimisation problem.

First, we set the initial state of the candidate gain matrix to the nominal value (i.e., $\widetilde{\mathbf{G}}^0 = \mathbf{G}_0$) and the temperature parameter T to 10. The temperature T will decrease in each iteration and finishes at 0.1. Then, we calculate the objective function $L(\mathbf{O}|\widetilde{\mathbf{G}}^0)$ and denote its value as L^0.

In each following round t, we propose a new candidate gain matrix by adding a random perturbation to the previous state:

$$\widetilde{\mathbf{G}}^t = \widetilde{\mathbf{G}}^{t-1} + \text{round}(\sigma^t \mathbf{R}^t) \tag{8}$$

Here, $\mathbf{R}^t$ is a 3-by-3 matrix that contains random floating-point values sampled from a standard uniform distribution at round t. We also use a step parameter σ^t to control the step size at each round; its initial state is set to 5 (i.e., $\sigma^0 = 5$) to allow bigger steps in the beginning.

Then, we calculate the objective value L^t and compare it with the previous objective value L^{t-1}. If L^t is lower than L^{t-1}, we always accept the proposal and keep $\widetilde{\mathbf{G}}^t$ as the latest estimate of gain matrix. Otherwise, we choose to accept $\widetilde{\mathbf{G}}^t$ or keep $\widetilde{\mathbf{G}}^{t-1}$ probabilistically to prevent getting stuck in a local minimum. If the proposal is accepted, we also keep the corresponding object value (i.e., $L^t = L^{t-1}$) and increase the step size slightly (e.g., $\sigma^{t+1} = \sigma^t \times 1.05$) to allow faster exploration. Otherwise, the step size will be decreased slightly (e.g., $\sigma^{t+1} = \sigma^t/1.05$) to help finding the minimum. We also set a lower bound for σ^t at 0.7 to prevent the step size being too small to update the candidate gain matrix in Equation 8.

3) Result: We test the algorithm on our iPhone XS test handset; the GyroID of this iPhone XS is shown in Equation 4. In particular, we first collect 50K gyroscope samples from the device at 200 Hz when it is stationary. We run our algorithm on this data for 5K iterations and present the result in Fig. 7. The figure shows the estimated GyroID stabilises after round 3383; the stabilised GyroID is the same as the one we estimated before noise was added. We have also tested the algorithm on an iPhone X and we were also able to recover the exact gain matrix using the same approach and setups.

4) Discussion: The experiments show that Apple's fix is still susceptible to probability-based attacks. However, we found that the attacker would need at least 50K data samples. Since the sample frequency of the gyroscope in recent iOS devices is 200 Hz, this means the attacker would need to collect gyroscope output for at least 4.2 minutes when the device is resting on a platform. In addition, the attack we proposed is also computation-intensive and thus it is unlikely to be implemented directly inside a mobile app. These restrictions make the attack less practical. Since Apple has also removed access to motion sensors from Safari and Webkit such an attack can now only be conducted via an app.

Even if Apple had adopted our proposed mitigation (adding uniform noise in the range $[-0.5, 0.5]$ to ADC outputs) this maximum likelihood estimation based attack would still work. However, the attacker would need even more samples. A better-designed noise scheme may enhance security further, but it might be harder to implement in mobile devices and could degrade the user experience.

VII. Related Work

Device fingerprinting is an important technique 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, 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 and Safari, 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 [16].

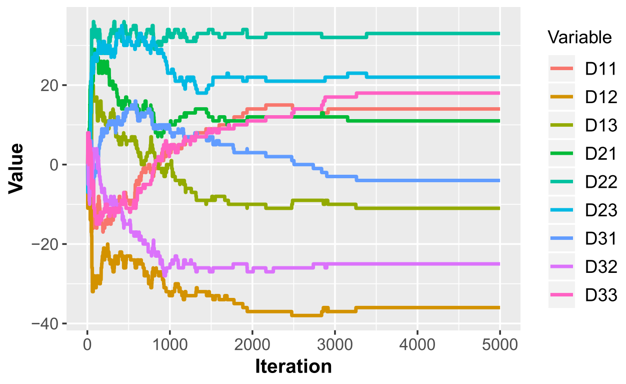


Fig. 7. Estimated GyroID at each iteration (iPhone XS).

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 [17]. 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 [18] and Panorama [19], can capture these malicious behaviours and report them to users.

A. Passive Device Fingerprinting

Passive device fingerprinting is the action of characterizing a target device by observing its network traffic. It analyses 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 behaviour in their traffic [20]. 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 [21]. 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.

B. 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 [12] is a popular browser fingerprinting library that utilises the characteristics of a browser, including the user-agent, version, plugins, fonts, and canvas. Apple has realised the risk of browser fingerprinting. From macOS Mojave, Safari scrubs most distinctive browser data, exposing only generic configuration information and default fonts [22]. 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.

C. Hardware Fingerprinting

Hardware fingerprints are generally consistent because it is typically 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 a smartphone and does not require any permission from users. Hardware modules that have been studied for fingerprinting purposes include: RF modules [23], [24], motion sensors [25], [26], clocks [27], camera [28], [29], and acoustic components [30], [31]. In particular, a well-known hardware fingerprint in digital forensics is the Photo Response Non-Uniformity (PRNU) of digital cameras. The PRNU is a result of manufacturing imperfections and inhomogeneity of silicon wafers. The classic algorithm to estimate the PRNU was presented by Lukás *et al.* [32]. Similar to SENSORID, the PRNU itself is stable to environmental conditions and is likely to be globally unique. Nevertheless, the exact value of the PRNU cannot be extracted and the quality of the estimated PRNU is dependent on the imaging processing used in the camera. Recently, Ba *et al.* presented a protocol named ABC to authenticate smartphones using the PRNU of their built-in camera [33]. According to their study, the PRNU estimated from one photo alone can identify the source smartphone camera with high accuracy. However, their study only focuses on two device models with a single camera. It is unclear whether the image fusion process in multi-camera devices would degrade the performance. In addition, accessing the camera or photos would require explicit permission from the user and thus it is less practical. In addition to smartphones, hardware fingerprinting also has applications on other targets. In particular, Son *et al.* used the *power-on offset calibration* of the gyroscope embedded in a drone to serve as its identity [34]. However, this *power-on offset calibration* is not factory calibration. The calibrated offsets are dynamically calculated every time the drone is turned on. Thus, it changes over time and varies with temperature. By comparison, our work is the first to recover the factory calibration parameters that are digital values stored in persistent memory and do not change afterwards.

Existing hardware fingerprinting techniques are mostly based on machine learning approaches. Bojinov *et al.* demonstrated it is possible to fingerprint both the speakerphone-microphone system and the accelerometer using typical clustering approaches [26]. 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 [35]. To increase accuracy, they used inaudible sound to stimulate the motion sensors. As a countermeasure, they suggested to better calibrate the motion sensors. However, they did not realise 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 [36]. 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 making 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 behaviour without knowing the device model in advance. Most recently, Das *et al.* studied the sensor API usage in popular websites [14]. 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 realised the idiosyncrasies across different sensors, none of them, to the best of our knowledge, has exploited the factory calibration to form a device fingerprint; this paper fills the gap.

VIII. CONCLUSION

In this paper we introduced the factory calibration fingerprinting attack: a new method of fingerprinting devices with embedded motion 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 coprocessor 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 an 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 is very likely to be globally unique for iPhone 6S, does not change on factory reset or after a software update. For older generations of iOS devices, such as the iPhone 4S and iPad Mini, we can further extract the ACCID and use it to provide extra entropy. In addition to iOS devices, we also conducted a large study of popular Android device models in the market and found that all Google Pixel phones except for Pixel 1/1 XL can be fingerprinted by our attack. We estimate the SENSORID entropy for each vulnerable Pixel model and show that it provides approximately 57 bits of entropy for Pixel 4/4 XL.

Furthermore, we analysed Apple's fix to our attack and showed that it is still possible to extract the GYROID even after the fix, although doing so would require significantly more data and computation power. Since it is no longer possible to access the motion sensors in iOS browsers, the opportunity to launch this attack is restricted to installed apps.

The concept of a calibration fingerprint is widely applicable. Although this paper mainly targets the motion sensors found in mobile devices, we anticipate the factory 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 factory calibration fingerprinting attacks on other types of sensor.

ACKNOWLEDGMENT

The authors would like to thank Nokia Bell Labs for valuable discussions. The opinions, findings, and conclusions or recommendations expressed are those of the authors and do not necessarily reflect those of the funders. They also thank Matthew Hall, Stephan A. Kollmann, Diana A. Vasile, Ricardo Mendes, Andrew Rice, and Amanda Prorok for helpful discussion and insight. They also thank anonymous reviewers for their feedback on this article.

REFERENCES

[1] G. Hogben. (2017). *Changes to Device Identifiers in Android o*. [Online]. Available: https://android-developers.googleblog.com/2017/04/changes-to-device-ide%ntifiers-in.html
[2] J. Zhang, A. R. Beresford, and I. Sheret, "SensorID: Sensor calibration fingerprinting for smartphones," in *Proc. IEEE Symp. Secur. Privacy (SP)*, May 2019, pp. 638–655.
[3] S. Poddar, V. Kumar, and A. Kumar, "A comprehensive overview of inertial sensor calibration techniques," *J. Dyn. Syst., Meas., Control*, vol. 139, no. 1, Jan. 2017, Art. no. 011006.
[4] A. Grammenos, C. Mascolo, and J. Crowcroft, "You are sensing, but are you biased? a user unaided sensor calibration approach for mobile sensing," *Proc. ACM Interact., Mobile, Wearable Ubiquitous Technol. (IMWUT)*, vol. 2, no. 1, p. 11, 2018.
[5] D. Tedaldi, "IMU calibration without mechanical equipment," Ph.D. dissertation, Dept. Inf. Eng., Univ. Padova, Padua, Italy, 2013.
[6] T. Michel, P. Geneves, H. Fourati, and N. Layaida, "On attitude estimation with smartphones," in *Proc. IEEE Int. Conf. Pervas. Comput. Commun. (PerCom)*, Mar. 2017, pp. 267–275.
[7] D. Tedaldi, A. Pretto, and E. Menegatti, "A robust and easy to implement method for IMU calibration without external equipments," in *Proc. IEEE Int. Conf. Robot. Autom. (ICRA)*, May 2014, pp. 3042–3049.
[8] R. Wei *et al.*, "A research on calibration of low-precision MEMS inertial sensors," in *Proc. 25th Chin. Control Decis. Conf. (CCDC)*, May 2013, pp. 3243–3247.
[9] I. Frosio, F. Pedersini, and N. A. Borghese, "Autocalibration of MEMS accelerometers," *IEEE Trans. Instrum. Meas.*, vol. 58, no. 6, pp. 2034–2041, Jun. 2009.
[10] STMicroelectronics. (2010). *L3g4200d: Three Axis Digital Output Gyroscope*. [Online]. Available: https://www.elecrow.com/download/L3G4200_AN3393.pdf
[11] STMicroelectronics. (2009). *Lis331dlh: Mems Digital Output Motion Sensor*. [Online]. Available: http://www.st.com/resource/en/datasheet/lis331dlh.pdf
[12] Valve. (2018). *Modern & Flexible Browser Fingerprinting Library*. [Online]. Available: https://github.com/Valve/fingerprintjs2
[13] 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-ch%art-2017-7
[14] A. Das, G. Acar, N. Borisov, and A. Pradeep, "The Web's sixth sense: A study of scripts accessing smartphone sensors," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Jan. 2018, pp. 1515–1532.
[15] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, no. 4598, pp. 671–680, 1983.
[16] Wikipedia. (2002). *Privacy and Electronic Communications Directive 2002*. [Online]. Available: https://en.wikipedia.org/wiki/Privacy_and_Electronic_Communications_Dir%ective_2002
[17] W. Enck, D. Octeau, P. D. McDaniel, and S. Chaudhuri, "A study of Android application security," in *Proc. USENIX Secur. Symp.*, vol. 2, 2011, p. 2.
[18] W. Enck *et al.*, "TaintDroid: An information-flow tracking system for realtime privacy monitoring on smartphones," *ACM Trans. Comput. Syst.*, vol. 32, no. 2, pp. 1–29, Jun. 2014.
[19] H. Yin, D. Song, M. Egele, C. Kruegel, and E. Kirda, "Panorama: Capturing system-wide information flow for malware detection and analysis," in *Proc. 14th ACM Conf. Comput. Commun. Secur. - CCS*, 2007, pp. 116–127.
[20] 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 *Proc. IEEE Conf. Commun. Netw. Secur. (CNS)*, Oct. 2013, pp. 305–313.
[21] C. Neumann, O. Heen, and S. Onno, "An empirical study of passive 802.11 device fingerprinting," in *Proc. 32nd Int. Conf. Distrib. Comput. Syst. Workshops*, Jun. 2012, pp. 593–602.
[22] L. H. Newman. (2018). *Apple Just Made Safari the Good Privacy Browser*. [Online]. Available: https://www.wired.com/story/apple-safari-privacy-wwdc
[23] T. Kohno, A. Broido, and K. C. Claffy, "Remote physical device fingerprinting," *IEEE Trans. Depend. Sec. Comput.*, vol. 2, no. 2, pp. 93–108, Feb. 2005.
[24] V. Brik, S. Banerjee, M. Gruteser, and S. Oh, "Wireless device identification with radiometric signatures," in *Proc. 14th ACM Int. Conf. Mobile Comput. Netw. - MobiCom*, 2008, pp. 116–127.
[25] S. Dey, N. Roy, W. Xu, R. R. Choudhury, and S. Nelakuditi, "AccelPrint: Imperfections of accelerometers make smartphones trackable," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2014, pp. 1–16.
[26] H. Bojinov, Y. Michalevsky, G. Nakibly, and D. Boneh, "Mobile device identification via sensor fingerprinting," 2014, *arXiv:1408.1416*. [Online]. Available: http://arxiv.org/abs/1408.1416
[27] I. Sanchez-Rola, I. Santos, and D. Balzarotti, "Clock around the clock: Time-based device fingerprinting," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur.*, Jan. 2018, pp. 1502–1514.
[28] J. Fridrich, "Digital image forensics," *IEEE Signal Process. Mag.*, vol. 26, no. 2, pp. 26–37, Mar. 2009.
[29] D. Valsesia, G. Coluccia, T. Bianchi, and E. Magli, "Compressed fingerprint matching and camera identification via random projections," *IEEE Trans. Inf. Forensics Security*, vol. 10, no. 7, pp. 1472–1485, Jul. 2015.
[30] A. Das, N. Borisov, and M. Caesar, "Do you hear what i hear?: Fingerprinting smart devices through embedded acoustic components," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur. - CCS*, 2014, pp. 441–452.
[31] Z. Zhou, W. Diao, X. Liu, and K. Zhang, "Acoustic fingerprinting revisited: Generate stable device ID stealthily with inaudible sound," in *Proc. ACM SIGSAC Conf. Comput. Commun. Secur. - CCS*, 2014, pp. 429–440.
[32] J. Luka, J. Fridrich, and M. Goljan, "Digital camera identification from sensor pattern noise," *IEEE Trans. Inf. Forensics Security*, vol. 1, no. 2, pp. 205–214, Jun. 2006.
[33] Z. Ba, S. Piao, X. Fu, D. Koutsonikolas, A. Mohaisen, and K. Ren, "ABC: Enabling smartphone authentication with built-in camera," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2018, pp. 1–15.
[34] Y. Son, J. Noh, J. Choi, and Y. Kim, "GyrosFinger: Fingerprinting drones for location tracking based on the outputs of MEMS gyroscopes," *ACM Trans. Privacy Secur.*, vol. 21, no. 2, pp. 1–25, Feb. 2018.
[35] A. Das, N. Borisov, and M. Caesar, "Tracking mobile Web users through motion sensors: Attacks and defenses," in *Proc. Netw. Distrib. Syst. Secur. Symp.*, 2016, pp. 1–15.
[36] A. Das, N. Borisov, and E. Chou, "Every move you make: Exploring practical issues in smartphone motion sensor fingerprinting and countermeasures," *Proc. Privacy Enhancing Technol.*, vol. 2018, no. 1, pp. 88–108, Jan. 2018.