Andreas Kurtz*, Hugo Gascon, Tobias Becker, Konrad Rieck, and Felix Freiling

Fingerprinting Mobile Devices Using Personalized Configurations

Abstract: Recently, Apple removed access to various device hardware identifiers that were frequently misused by iOS third-party apps to track users. We are, therefore, now studying the extent to which users of smartphones can still be uniquely identified simply through their personalized device configurations. Using Apple's iOS as an example, we show how a device fingerprint can be computed using 29 different configuration features. These features can be queried from arbitrary third-party apps via the official SDK. Experimental evaluations based on almost 13,000 fingerprints from approximately 8,000 different real-world devices show that (1) *all* fingerprints are unique and distinguishable; and (2) utilizing a supervised learning approach allows returning users or their devices to be recognized with a total accuracy of 97% over time.

Keywords: Fingerprinting, Apple iOS, Mobile Device, Privacy

DOI 10.1515/popets-2015-0027
Received 2015-04-15; revised 2015-07-15; accepted 2015-07-15.

1 Introduction

As of mid-2014, more than 1.2 million apps have been made available in Apple's App Store—and downloaded more than 50 billion times [1]. Google's Play Store also contains over 1.3 million apps, including more than 1.1 million free apps [2]. Free apps often raise the question of their business model. If downloading an app costs nothing, users often become unwitting products, with app providers using advertising in an attempt to make the app financially viable.

In the past, various studies have shown that, in the case of Apple's App Store, around half of all the free apps contain advertising or tracking libraries [3–5]. It can be expected that apps from Google Play exhibit similar properties. These libraries observe user behavior and collect personal information about users. In some cases, they send this collected data to advertising networks—which allows in-app advertising to be targeted as accurately as possible—or sell the user profile. Although most users do not notice these background activities and, therefore, cannot opt out of them, their impact on user privacy is obvious. Advertising and tracking providers continuously receive comprehensive, partially cross-app profiles of app users' usage habits.

In order to match the collected data and activities to an individual user, the user's device must be uniquely identifiable, a task usually called *device fingerprinting.* It is well known that Android allows access to the unique device ID (IMEI for GSM and MEID/ESN for CDMA phones) which makes it almost trivial to perfectly fingerprint devices [6]. Until the release of iOS 7, various hardware identifiers were also available on Apple's mobile platform. In addition to the Unique Device Identifier (UDID), which is based, among other elements, on the device serial number, the WiFi MAC address was, in many cases, used to uniquely identify users or their devices [3, 7]. As a response to privacy concerns, Apple started to restrict access to hardware-based identifying features in 2013 [8]. App providers, therefore, have no longer been able to use hardware identifiers to recognize returning devices. As we show in this paper, however, this does not necessarily mean that devices cannot be fingerprinted.

1.1 Challenges

To the best of our knowledge, there has, until now, been no study focusing on fingerprinting mobile devices using personalized device configurations. This is a fundamentally different approach to previous work since it assumes that the user voluntarily installs software (the app) and that the app operates silently without the

***Corresponding Author: Andreas Kurtz:** Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU), E-mail: andreas.kurtz@cs.fau.de
Hugo Gascon: University of Göttingen, E-mail: hgascon@cs.uni-goettingen.de
Tobias Becker: Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU), E-mail: tobias_becker@me.com
Konrad Rieck: University of Göttingen, E-mail: konrad.rieck@uni-goettingen.de
Felix Freiling: Friedrich-Alexander-Universität Erlangen-Nürnberg (FAU), E-mail: felix.freiling@cs.fau.de

user's further cooperation (apart from regularly using the app for its intended purpose).

The biggest real-world challenge with all fingerprinting approaches is balancing out the fingerprint features with regards to *diversity* and *stability*. The more features that are used in creating a fingerprint, the more likely it is that all data records will be different—and therefore distinguishable. In practice, however, features change over time, particularly on mobile devices, which are used multiple times a day. In turn, the more features are used, the higher the probability that individual data fields will change, thus affecting the fingerprint's stability. In most cases, therefore, a high level of diversity in features reduces the stability. A high level of stability can often be achieved only by including as few features as possible. In a fingerprinting context, the trade-off between diversity and stability has to be mastered.

1.2 Contributions

In this paper, we present a purely software-based approach to fingerprinting mobile devices based on their personalized configurations, and point out the resulting privacy risks. Instead of focusing on web browsers, which were the main objects of interest in the past when it came to client fingerprinting, we take the viewpoint of a third-party app that can collect information about the entire device configuration. Since Apple iOS is currently the platform with the most preventive mechanisms, iOS was deliberately chosen for our study. It is, however, clear that our findings can be similarly applied to other platforms.

First, we systematically searched the iOS SDK for uniquely distinguishable (=identifying) fingerprint features. This process revealed 29 features, which were then added to our fingerprinting algorithm. These include, for example, device names, language settings, lists of apps installed and most played songs. Finally, we created our own app that combined these features and, if the user gave permission, sent the data to our server for evaluation. Any personally identifiable data was anonymized before transmission using hashing. The app was then placed in the App Store for download.

During our 140-day study period, we collected almost 13,000 data records from 8,000 different devices. All the fingerprints we discovered were unique. Although the fingerprints are clearly distinguishable in theory, it is hard to use them for long-term tracking of users. In practice, individual fingerprint features change over time as the device is used. This aspect was also investigated, as almost 57% of the data records transmitted to us came from recurring devices. This enabled us to observe the change in data over time and to determine criteria for feature stability.

Following this, we propose a robust solution for measuring the general similarity between any pair of fingerprints, independent of their size and structure. We then determine an optimal similarity threshold using a supervised learning approach that considers the chronological order in which fingerprints would be received in a real world scenario. This approach enables us to uniquely identify devices with a total accuracy of 93.76% when all 29 features are included.

We then evaluate the collected data from various different perspectives and gradually reduce the feature space to determine features or combinations that would lead to an accuracy increase. Some test cases, for example, use only the list of installed apps or the top 50 most-played songs. Both pieces of information are freely available to third-party apps and querying them is very unobtrusive. Our approach proved capable of uniquely identifying devices purely on the basis of the apps installed, with an overall accuracy of more than 97%. Moreover, identifying devices based solely on the user's music taste succeeded with a total accuracy of 94.20%. Finally, we discuss countermeasures and demonstrate how identification accuracy could be drastically decreased if Apple further tightened the app sandbox to prevent unrestricted access to only a few strong distinguishing features.

Summarizing, our main contributions are:

1. We present a novel and highly accurate software-based approach to mobile device fingerprinting that relies solely on correlating information provided by the mobile operating system. We provide a taxonomy of mobile fingerprint data sources and describe a set of 29 identifying properties within Apple's iOS platform. Moreover, we describe the concept of a data collection app that was made available in the App Store to collect these properties from real-world devices.
2. We formulate the question of whether or not fingerprints originate from the same device as a multi-class classification problem. We then show that this challenge can be accurately solved using a simple threshold-based classifier based on fingerprint similarity.

3. We investigate the accuracy and robustness of our approach by evaluating around 13,000 fingerprint data records from nearly 8,000 different real-world devices. The experimental results show our method to be effective in identifying mobile devices with high accuracy in a low-dimensional feature space.

2 Related Work

In his seminal work, Eckersley [9] pioneered the area of *browser fingerprinting* and created fingerprints using the browser version reported by the user agent and/or the browser configuration (plugins, fonts etc.). He was then able to use the fingerprints created using this method to recognize returning browsers, even if their individual features had changed over time. The false positive rate was only 0.87%. This sparked several follow-up studies, investigating new browser fingerprinting methods or examining how browser fingerprinting was used "in the wild" for user tracking [10–13]. In principle, techniques for browser fingerprinting can also be used on mobile devices. Specialized apps have, however, replaced generic browsers for many usage scenarios. Furthermore, mobile browsers are barely customizable (no plugins, no fonts, almost no differences between versions) and therefore do not have as many distinguishing characteristics as desktop browsers. Even Eckersley admits that his approach is not suitable for use on mobile devices, as "iPhone and Android browsers [are] significantly more uniform and harder to fingerprint than desktop browsers" [9].

Therefore, in the past, alternative methods for *mobile device fingerprinting* have been proposed. These approaches evaluate different (mostly physical) characteristics and draw inferences about the device platform. Classic examples include methods for identifying image sensors in digital cameras based on various aspects of their processing pipelines, like the photo-response non-uniformity (PRNU) of CCD sensors [14, 15]. While this can also be used to identify smartphones (based on their built-in cameras), these calculations are extremely resource-intensive and require a large number of specially crafted images [16, 17].

Other work on mobile device fingerprinting also focuses on fingerprinting device sensors. Previous studies have, for example, investigated embedded acoustic components [18–20]. The frequency response of the speaker-microphone system was measured by playing sound over the speaker and simultaneously recording it using the microphone. Due to differences identified in their frequency responses, the authors were able to identify 95% of 17 devices [18], or to match 98% of the audio clips recorded to 50 different Android smartphones [19]. Other sensor-based processes were used to analyze the data reported by the accelerometer. The accuracy rates were 53% [18], or 96% [21]. As all these methods require user cooperation, they cannot be used as easily and silently as browser fingerprints.

In previous studies, information provided by mobile SDKs to third-party apps has been used for various different purposes. For instance, the list of installed apps has been successfully used to identify user traits [22]. Properties like running apps, the device model, or operating system type have also been used within collaborative approaches to diagnose energy anomalies [23]. Although these approaches could obviously affect user privacy in various ways, they are not directly related to our objective—that of fingerprinting devices for long-term user tracking purposes.

3 Background

In this section, we lay out the history of identifiers in Apple's iOS platform.

Until recently, third-party apps on Apple's platforms could make use of various hardware identifiers (UDID, WiFi MAC), enabling them to uniquely identify devices. As these features were often misused to match personal data and behavior patterns to individual users and to transmit this information to advertising networks, Apple reacted by eliminating access to these hardware features with the release of iOS 7.

Even before the hardware identifiers were removed, two alternative software identifiers designed to be used for ad tracking were introduced in iOS 6. These two non-personal, non-permanent identifiers are designed to provide better privacy protection: the Advertising Identifier (IDFA) is an alphanumeric string which is unique to each device, while the Identifier For Vendor (IDFV) is identical only for apps from the same provider. Both identifiers are suitable for user tracking only to a limited extent. On the one hand, the IDFV is not identical across apps and, although both identifiers are persisted by the system, the values can be reset as desired. The IDFV is, for example, automatically reset if all apps from a provider are uninstalled, while the IDFA can be manually reset via the Settings app. The identifiers are also reset if a device restore is carried out. In addition, Apple emphasizes that misusing the identifiers will lead to the app being rejected during the App Store review process [24]. Neither identifier, therefore, fulfills the requirements of a *global identifier* [9], which remains constant over time and enables tracking without the user either noticing or being able to opt out.

It is worth noting that our approach does not suffer from these drawbacks, as our method involves fingerprinting a user's device configuration. As long as their personal configurations do not change significantly, users remain identifiable in the long-term to third-party apps, even if they change their devices. Moreover, as all required information is provided by the official SDK, the App Store review process does not state any limitation as shown below.

It should also be noted that, in the past, several alternatives were proposed in response to Apple's removal of hardware identifiers. In response to the deprecation of the official UDID, the *OpenUDID* project provided a "persistent and sufficiently unique" identifier that was based on a randomly generated universally unique identifier (UUID). This was then shared across apps through private pasteboards [25]. This unofficial data exchange mechanism was, however, disabled by Apple with the release of iOS 7 [26]. Following this, in response to Apple's introduction of the IDFA (which may, according to Apple, be used only for advertising purposes), the OpenIDFA project proposed an alternative tracking identifier [27]. Like our approach, the *OpenIDFA* identifier factors in various device and operating system properties, including the device model, the iOS version, the preferred language and the device boot time. From these values, an SHA-256 hash is computed and converted to a UUID-like format. The identifier generated in this way is designed to persist only for a single day, but its validity may be extended for up to three days if required. If any of the included properties change earlier, however, the identifier value is also instantly changed. The OpenIDFA cannot, therefore, be used for long-term tracking. According to its authors, this is not a limitation: instead, it is a conscious decision in favor of user privacy.

4 Methodology

In this section, we describe the overall methodology that we used to determine whether or not today's mobile device configurations are so personalized that they create unique fingerprints—which can then be used for tracking purposes.

To do this, in Section 4.1, we describe the various data source classes from which features can be extracted for configuration-based mobile device fingerprinting. From these classes, we then extract 29 features that we considered suitable for fingerprinting Apple iOS devices. We then describe our app, which was made available in the App Store in order to collect this feature data from a large sample of real-world devices. Finally, in Section 4.2, we introduce our evaluation approaches, focusing on the distance metric we used to measure fingerprint similarity.

4.1 Data Collection

One method of fingerprinting mobile devices that has gone unresearched until now involves correlating information provided by the operating system and queried by third-party apps via the SDK. This method distinguishes between device-dependent and person-related data. As both categories are becoming increasingly blurred on highly personalized mobile devices, however, we distinguish data in the following sections by the context in which it is made available.

1. *Contextual Data*: Information in this category is strongly context-dependent and thus is subject to frequent change. This category includes, for example, location data reported by the GPS sensor. In order to use this data to identify users or their devices, it must first be collected over a longer period of time and then be evaluated. As this type of context-dependent data is usually only collected when the app is actively used, it is of limited use for fingerprinting, particularly when apps are rarely used.
2. *Non-Contextual Data*: Non-contextual data can be divided into *constant* and *time-variant* data. Whereas constant data (such as the device model) does not change over the lifetime of a device, time-variant data (such as the operating system version) can have slightly different values over time. The frequency of the changes is, however, minor in comparison to contextual data. As non-contextual data is complete at the time of the query, rather than having to be observed over a longer period of time, data from this second category is particularly suitable for software-based fingerprinting. In the following section, we will, therefore, focus exclusively on non-contextual data.

4.1.1 iOS Fingerprint Features

The iOS SDK comprises four levels with a total of 57 frameworks for the most diverse purposes. All functions of these frameworks are grouped by level and described by Apple in the iOS Developer Library [28]. We systematically searched these documents for non-contextual identifying features, eliminating frameworks that obviously contained neither user- nor device-related data, such as OpenGL, CoreGraphics, MapKit, etc.

Since the release of iOS 6, some of the data accessible via the SDK (personal user information such as calendar and contact entries) can be accessed only if the user gives his or her consent during the first access attempt. Generally speaking, data in mobile SDKs can, therefore, be categorized by its context, as well as by its availability. We chose the following categorization:

Public Resources
Data in this category can be queried without the user's permission or knowledge. Most of this information relates to the device or its configuration, including, for example, the device model, the current iOS version, the current time zone, installed keyboards, carrier name, device name, etc. It should be noted that the device name is freely configurable, but is configured using the user's first and last names when a new iOS device is set up (e.g. *<first name><last name>*'s iPhone). Moreover, this category also contains personal information, including the user's media library. As information on the song titles stored on the device can be requested by any app unnoticed, the user's music taste can be analyzed and used as a fingerprinting feature. For this, we retrieved the play count of each song from the full list of each user's music collection and determined the top 50 most-played songs. We also included the apps installed, as we expected them to be highly personalized. As iOS provides no official API for accessing this information, alternative approaches have emerged in the past. These determine installed apps by checking whether or not certain app-specific URL schemes are supported (*canOpenURL*). It should be noted that this leads only to a subset of available apps, depending on the number of available apps that provide custom URL schemes and the size and quality of the URL scheme database. We drew on the URL scheme database provided by handleopenurl.com.

Protected Resources
Data in this category can be accessed only once the user has given an app permission to access a protected resource during its first attempt. iOS protected resources include location data, photos, contacts, calendar data, reminders, sensor data (microphone, camera etc.), and social network accounts. As a rule, this protected information is only partly suitable for use in fingerprinting, as it cannot be queried freely by every app. If an app has received permission to access protected resources, however (for example, if a messaging app has been allowed to access a user's contacts), this can have a pronounced effect on the uniqueness of the fingerprint determined from it. In order to study their actual influence, we have, to a certain degree, included the calendar, reminders, photo library, contacts and social network accounts in our studies. In doing so, we set the following restrictions: as calendar and reminder entries often change, we have taken into account only the superior calendar names and the names of reminder lists. In the photo library, due to the volume of data expected, we limited our evaluations to the metadata in the album titles. For social networking accounts, we did not include Facebook, as the stored usernames can be queried only with a registered Facebook API key.

File System
In addition to all SDK-based information, we discovered an additional feature source, the file system. Strictly speaking, this is a special area of Public Resources. Under iOS, the app sandbox is actually designed to prevent apps from accessing files outside their containers. When analyzing the Sandbox Profile, however, we identified numerous exceptions to this rule which enable any app to access key iOS configuration files, unnoticed and without permission. The information contained therein could also be used for fingerprinting. Later analyses showed that apps which could access these files were not rejected during the App Store approval process and that, consequently, App Store apps could also be used to access them without restriction (see Section 4.1).

Via various files in the Preferences directory, the Apple ID of a user (personal email address), the Game Center Player ID, or the Code Signing Identity can be obtained. As this data is strongly person-related (or, for fingerprinting, clearly distinguishing), and can be read unnoticed by any app via a simple file system access, we informed Apple of this sandbox defect. It should be noted that these unique features were not taken into account in our fingerprinting approach, as users would be up to 100% identifiable. With the release of iOS 8 in September 2014, these defects were remedied and access to the iOS Configurations directory was restricted (CVE-2014-4362). Thus, user privacy was significantly improved by our study.

Any third-party app can, however, still access a central icon cache folder under iOS. This folder acts as temporary storage for the icons belonging to the installed apps. As the icon filenames contain the respective bundle identifiers of the apps, enumerating the directory's contents can relatively easily produce a full list of the apps installed on the system. This finding has not been addressed by Apple, although we pointed out the resulting effects on user privacy. It should be noted that, in comparison to the previously introduced approach to determine installed apps by iterating over available custom URL schemes, this file-system-based method provides any app with not only a subset, but a full list of all installed apps with a single directory access.

At the end, we determined 29 different features from all data sources. These were found suitable for use in fingerprinting. A complete listing of all the features collected is contained in Appendix A.

4.1.2 iOS App Store App

In order to collect features from as many different devices as possible, we created an iOS app and made it available in the App Store. Essentially, the app was responsible for collecting the fingerprint features and sending them to our server. In doing so, we made it a priority to inform users clearly about the background and aims of our study, about how the data collected would be used, and about each step taken. In order to do this, the app provides a wizard that takes users through the individual steps and provides detailed explanations of each. The app was offered in multiple languages, including English. Essentially, the wizard guided users through the following three phases:

1. *Background Information:* At the beginning, two pages of background information explained our study and our fingerprinting approach.
2. *Data Collection:* After that, the actual data collection was performed. For this to proceed, users had to explicitly launch the collection process.
   (a) In the first step, all the information freely available via the SDK and the file system was collected. Once this process was completed, the user was informed.
   (b) The next step involved accessing protected resources. This step was optional and could be skipped. If users wanted to let our app access protected resources, they were able to continue the collection process and then agree to the iOS authorization checks.

   Once the collection process was completed, all personal information, including the user's Apple ID, the apps installed, contact entries, song titles, etc, was anonymized using a cryptographic hash. This preserved user privacy at all times, while still enabling us to evaluate the data for distinguishing features.

   Finally, the data collected was displayed in tabular form in the app. This enabled the user to view all the data and to check that all personally identifiable information had been suitably anonymized.
3. *Data Delivery and Results:* In the final step, the user was able to voluntarily send the data to our server in order to participate in the study. The data was sent to a web service in JSON format. There, the transmitted data records were compared with all the other data records received. The user was then informed of the number of fingerprints available and told whether or not his or her fingerprint was unique, or distinguishable from the others. The user was also directly informed of the features which made his or her fingerprint unique. These features were highlighted in color in the data overview. This interaction was intended both to provide participants with an incentive to take part in our study and to inform them of the problem of fingerprinting. This is similar to Eckersley's study on web browsers [9].

4.1.3 Identification of Recurring Devices

In order to observe the changes in individual features over time—and thus to determine the feature stability—we had to find a way to match the data records received unambiguously to a single device. To do this, we generated a random, alphanumeric, unique identifier (cookie) whenever a data record was received by the server. We then stored the data record and the cookie together in the backend database and reported the cookie's value back to the app. The cookie was then appended to all subsequent fingerprints received from that device. This enabled us to match them appropriately.

We stored the cookie inside the iOS keychain on the device. The keychain is a storage location provided by iOS for encrypting and storing sensitive data. One feature of the keychain is that entries placed there by an app remain on the device, even if the app is removed. This meant that our cookie was permanently stored on the devices, enabling us to unambiguously match data records sent to us from recurring devices, even if the app had been uninstalled and then reinstalled. It should be noted that such a cookie is specific to this particular app only (or apps from a single vendor) and therefore does not fulfill the requirements of a global identifier that allows identification across apps.

In order to remind users regularly about our study, we included a mechanism within the app for push notifications. If users agreed to receive notifications from our app, we reminded them once a week to relaunch the app and send us an updated data record. This method enabled us to reliably observe changes in individual features over time.

4.2 Fingerprint Similarity

To evaluate whether a set of fingerprints can be effectively linked to an individual device, we need to quantify how fingerprints differ from each other. Although it is clear that the distance between two identical fingerprints should be zero, fingerprints are composed of a set of heterogeneous data, making it far from trivial to design a method to measure their similarity.

Fingerprints are formed by different types of features: Boolean properties, strings and lists. A list may be also a tree-like structure, such as that used for contacts or songs, where each item contains several strings, or lists of strings as sub-properties (e.g. work phone number, personal email, social media accounts, etc.). Although the comparison of Boolean values can be done on a one-to-one basis, strings and lists allow for partial matches. For example, two devices belonging to users who know each other may share several items on their contact lists. The similarity of such larger properties that induce partial or inexact matches should also be quantified to account for more granular differences.

To solve this problem, we propose a solution for measuring the general similarity between any pair of fingerprints, independent of their size and structure. First, we begin by *flattening* the structure of the fingerprint. If we represent the set of features as a shallow tree, flattening is equivalent to transforming every fingerprint into a set of paths from a unique root to all the leaves.

Each fingerprint then has an arbitrary size and is represented by a set of strings with the form *name_subname_..._value*, as shown in the following example:

```
voiceOver_False
model_iPhone6,2
top50Songs_album_27333bed7d8f5575c51c3a067c0c285b
contacts_firstname_181ff5eb6d184d261d6b28480110e599
[..]
```

To illustrate this, let us consider two users who share the same contact in their address book. One of them has included the contact's middle name, while the other has not. If we evaluate a contact as a unique feature, the two items will not match. By flattening each value, an overall similarity score can be computed based on common values. In this example, the contacts will match to a certain degree although the entries are not exactly identical. This approach allows us to account for more subtle differences by adding some fuzziness to the representation of the fingerprint.

Once fingerprints are flattened, we make use of the *Jaccard similarity coefficient* to compute the similarity between each one of the sets:

$$J(A,B) = \frac{|A \cap B|}{|A \cup B|} \tag{1}$$

$$0 \leq J(A,B) \leq 1 \tag{2}$$

This index measures the similarity between two sets, A and B, and takes values between 0 and 1. It is computed as the quotient of the intersection of both sets and their joint set. In this way, we are able to compute a similarity score between 0 and 1 for each pair of fingerprints in our dataset.

Several remarks can be made here. Firstly, in our implementation of the Jaccard index, the multiset resulting from flattening each fingerprint is transformed into a set. This means that the similarity measure will be more influenced by shared properties and less by the amount of such properties. For example, if two fingerprints include contacts with Facebook accounts, the similarity will be influenced by the user names that match and not by the number of contacts with Facebook accounts. Secondly, the flattening process discards some information from each fingerprint. This strategy does, however, allow us to represent fingerprints as sets of manageable size whose similarity can be measured very fast and memory-efficient. As we discuss in Section 5.5, our method results in better performance than more advanced techniques, despite its low complexity.

5 Evaluation

This section describes the results of our evaluation. All the results are based on real-world data collected using our App Store app.

5.1 Sample Data

Once our app was made available in the App Store, we advertised our study in social networks and various mailing lists, asking for volunteers to take part. Several news portals also reported on our study, thus increasing the app's popularity more rapidly. At one point, it achieved first position in the Utilities category of the App Store's Top App Charts.

As the app developed a certain life of its own via the App Store charts and the detailed explanations provided in the app were also comprehensible to less technically experienced users, we assume that our data is only slightly (if at all) biased towards privacy-conscious users and is primarily representative.

In total, our app was downloaded over 16,800 times during our study period of 140 days between May and September 2014 (14,100 times via iPhone, 1,830 via iPad, and 165 via the iPod Touch). During this period, 12,937 fingerprints were submitted from 7,815 different devices (cookies). 2,261 devices delivered multiple fingerprints (about 29% recurring devices). 7,383 of the fingerprints submitted (about 57%) came from those recurring devices. Almost two-thirds of all users also granted our app access to protected resources.

5.2 Fingerprint Diversity & Stability

One of the basic prerequisites for fingerprinting is that all data records must differ from each other. Therefore, there should be no exact matches for fingerprints from different devices. In the first step, we verified this precondition and determined that, at the time of the evaluation, all the data records we had hitherto received were unique and therefore distinguishable. Among the 12,937 fingerprints, no two were identical.

To evaluate how much fingerprints changed over time, we grouped all the available records by device and studied each group containing more than one data record in more detail. Figure 1 shows the percentage of the average fingerprint changes over time. Each data point represents the average amount of fingerprint changes per day. The small, weekly changes can be attributed to the fact that we sent users a weekly push notification asking them to submit a new fingerprint. However, first and foremost, the graph shows that, on average, fingerprints change by less than 10% per month.

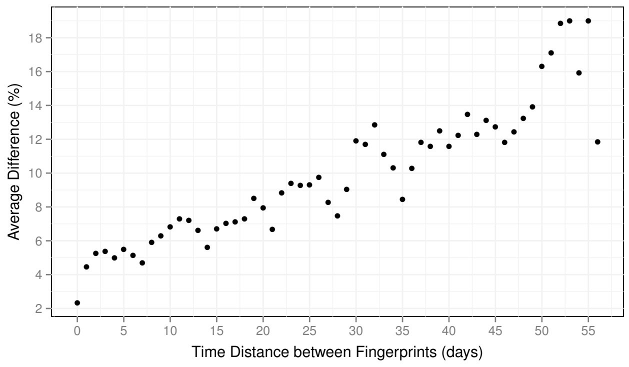


Fig. 1. Average Fingerprint Changes over Time

At first glance, this seems surprising, particularly when considering that mobile devices are closely integrated in our daily routines and are therefore heavily used, often multiple times a day. This low level of fluctuation confirmed our initial assumption that personalized configurations might be well suited for fingerprinting purposes.

In the next step, we determined how often individual features change. For this, we took the group of fingerprints from recurring devices and determined how often each feature changed over time. This information is later required to balance the diversity/stability trade-off by choosing suitable feature combinations. A heatmap visualizing the change ratio over time is shown in Figure 2. A full listing, including the exact stability values, is provided in Appendix A.

Using this method, we determined that features like the public IP address and the Internet Service Provider changed, as expected, very frequently. A user's installed apps, the top 50 songs and the WiFi SSID also often changed, but turned out to be slightly more stable. We also noticed that, around 40 days after the publication of our app, the iOS version reported by the users changed significantly. This was due to the fact that Apple released an iOS update at precisely that time.

We then used these changes to calculate the stability for every feature. The most stable features were configuration settings, like, for example, assistive features in iOS for people with disabilities. The setting which determines whether or not VoIP is permitted did also change only on a few devices. This is hardly surprising, as this setting is set by the carrier. Although these configuration features turned out to be most sta-

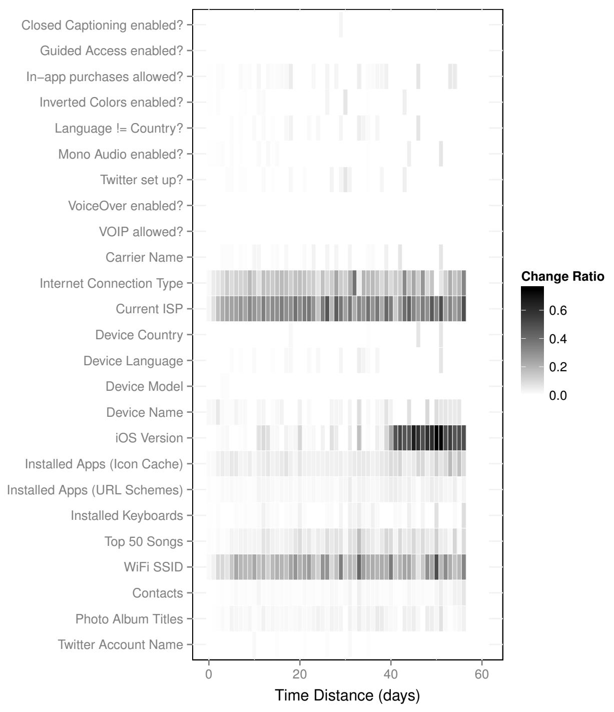


Fig. 2. Average Fingerprint Changes per Feature

ble, they were Boolean values and therefore carry less information. Among the features with higher information content, the top 50 songs, the list of installed apps, the device name and the WiFi SSID were the most stable public resources. Among the protected resources, the Twitter account name and reminder/calendar list names and photo album titles were found to be most stable.

5.3 Experimental Setup

After we determined that, in theory, all fingerprints are distinguishable from one another, we studied the extent to which it could practically be determined that two or more fingerprints came from the same device. As has already been mentioned, matching is no easy task, as a device's fingerprint changes over time.

The decision as to whether two fingerprints originate from the same device is made on the basis of their similarity (see Section 4.2). More specifically, we try to answer two questions:

1. How is a fingerprint identified as the first one submitted from a previously unknown device?
2. How is a fingerprint matched to a device whose fingerprints are already in the database?

As mentioned in Section 4.2, fingerprints submitted from the same device at different points in time tend, in general, to be more similar to each other than fingerprints from different devices. In certain cases, however, they might also be closer to fingerprints from other devices. Therefore our goal is to find an optimal similarity threshold between fingerprints from the same device that maximizes the accuracy on both types of decisions. We formalize the problem by introducing two performance metrics:

1. **Discrimination**: the ability to correctly assign a new label to the first fingerprint F submitted from a device.
2. **Re-identification**: the ability to correctly assign the label of a device's first fingerprint to a new fingerprint F submitted from the same device later in time.

In the same way that the threshold of a binary classifier determines a trade-off between true positive and false negative decisions, it can be seen that a trade-off also exists between these two metrics. If the threshold is set to 1, a new label will be assigned to each fingerprint and discrimination and re-identification accuracy will be 1 and 0 respectively. On the contrary, if the threshold is set to 0, every fingerprint will be assigned to the same class, making discrimination and re-identification accuracy 0 and 1 in this case. As a result, the higher the accuracy is when fingerprinting a device, the lower the accuracy is when fingerprinting the device again, and vice versa.

Finding the threshold that maximizes the total accuracy (i.e. the percentage of correct decisions) can be posed as a multi-class classification problem. Nevertheless—and in order to design an experiment that resembles the condition of a real attack—we must be careful in selecting a suitable learning approach. Unsupervised approaches, for instance, may suit our problem but, unfortunately, standard clustering algorithms like *k-means* are not designed to generate new clusters as new classes of data are received in an ongoing process.

We therefore propose a supervised learning approach that does not disrupt the chronological order and allows us to make correct decisions during testing, even if data from unlabeled devices is received.

We proceed in the following manner. First, we split the dataset into training and testing sets (66%/34%) without modifying their chronological order. Then, we evaluate the performance of our classification algorithm on the split of training data across the parameter space. The discrimination, re-identification and total accuracy

Feature Set	Threshold	Training Acc.	Testing Acc.	Discr. Acc.	Re-ident. Acc.
Installed Apps (URL) & Device Name	0.86	98.86%	97.86%	99.00%	96.67%
Installed Apps (Icon Cache)	0.78	98.89%	97.78%	98.28%	97.25%
Public Resources	0.77	98.85%	95.65%	98.91%	92.07%
Protected Resources	0.94	97.14%	95.15%	95.52%	94.74%
Top 50 Songs	0.74	98.09%	94.20%	97.29%	90.87%
All	0.73	95.63%	93.76%	96.83%	90.56%
Device Name & WiFi	0.99	90.17%	78.43%	96.97%	59.20%
Public w/o Apps, Name, Songs	0.83	88.01%	70.90%	96.24%	44.60%
Public w/o Apps, Name, Songs, WiFi	0.99	86.93%	58.78%	98.37%	17.70%

Table 1. Comparison of different feature set combinations (ordered by total testing accuracy).

are computed by finding the closest fingerprint F_2 to each fingerprint F_1 from those in the training set belonging to different devices and those from the same device that were submitted previously in time. The label of F_2 is assigned to F_1 if their similarity is above the threshold. Otherwise, a new label is assigned to F_1. During training the similarity threshold takes a range of 100 values between 0 and 1. The threshold producing the highest total accuracy is evaluated on the split of test data.

5.4 Fingerprinting Results

In an initial test run, we included all features (public and protected resources) in our analysis. In doing so, we achieved a top *training accuracy* of 95.63%, at a threshold value of 0.73. It should be noted that the training accuracy is an indicator for how well a learning method can perform in the best possible case, while the *testing accuracy* provides the performance likely to be observed in practice. Figure 3a shows the evolution of the training accuracy in relation to the threshold for the combinations of features with best performance as presented in Table 1.

Using all features combined, the best threshold found during training achieved a testing accuracy of 93.76%. This metric represents the total percentage of fingerprints that have been correctly assigned to their original device. As described in Section 5.3, we can also measure the performance of the classifier in two special cases: If a fingerprint received is the first one submitted from a device, or if more fingerprints from this device have previously been received.

More specifically, every time a hitherto unknown device submits a fingerprint we want to be able to identify that this is indeed a new device. When we use all features available, we are able to achieve this for almost all devices. In particular, the discrimination accuracy was 96.83%.

If a fingerprint is not the first one submitted by a device, we want to recognize this fact and be able to assign the fingerprint to the same device class. In 90.56% of the cases, fingerprints from recurring devices were correctly assigned as indicated by the re- identification accuracy.

As previously explained, the threshold parameter was optimized during training with the total accuracy as the target. However, a trade-off between discrimination and re-identification exists. Figure 3b shows the relation between the two metrics during training. The optimal operating point of any classifier is located at $(1, 1)$. The best possible performance point for each combination of features will, therefore, be achieved at the point in the curve closest to this optimal point.

In the next step, we divided the feature space and examined whether or not protected resources achieve better performance than public resources. Our hypothesis was that, as users must confirm access to protected resources, the data should be more strongly personalized and therefore provide better results. Surprisingly, we determined that the differentiation played almost no role. In both cases, we measured a similarly high total accuracy rate of around 95%. The discrimination was slightly better for public resources (98.91% vs. 95.52%), whereas the re-identification accuracy was higher for protected resources (94.74% vs. 92.25%). This led us to the conclusion that our fingerprinting approach achieves accurate results, irrespective of whether or not an app receives access to protected resources.

In further tests, we reduced the feature space even more and examined which public resources or feature combinations achieved the best performance. Based on our previous tests on feature stability analysis, in the first step, we used only the complete list of installed apps (icon cache). In doing so, we achieved our hitherto best accuracy of 97.78%. However, when we then added the device name to the list of all apps, the total accuracy decreased, with re-identification accuracy being reduced by around 2%. Following this, we used a subset of the installed apps, namely those which can be determined using URL schemes. In this case, the total accuracy rate was only around 94%. Combining this feature vector with the device name, however, resulted in the highest total accuracy rating of all our tests (97.86%). This feature combination enabled a discrimination accuracy of 99.00% and a re-identification accuracy of 96.67%.

Finally, we investigated the extent to which the top 50 songs (or a user's music taste) serves as a suitable fingerprinting criterion. Using this criterion still enabled us to achieve a total accuracy of 94.20%. The accuracy decreased considerably if we combined only the device name and the Wifi SSID (used alone, neither feature yielded any useful results). Using this feature combination, however, enabled us to measure an accuracy of 78.43%. Whereas the discrimination accuracy was still relatively high (more than 96%), this combination failed for re-identification—just over half of all fingerprints (59.20%) were correctly assigned.

5.5 Classifier Comparison

As no similar method for this problem exists in the literature, we compare our method in this section with a baseline random classifier and a standard (but more advanced) machine learning-based approach based on *support vector machines*.

Due to its simplicity, a random classifier is usually established as a baseline for performance. In our implementation, we assign each fingerprint in the dataset to a random device while keeping the original label distribution. Table 2 shows the accuracy of this type of baseline classifier when evaluated on our complete dataset. The low performance values obtained reflect the non- triviality of the problem and highlight the good performance of our method.

Accuracy	Discrimination Acc.	Re-idententification Acc.
7.83×10^{-5}%	1.29×10^{-4}%	0%

Table 2. Accuracy of a random classifier on the complete dataset.

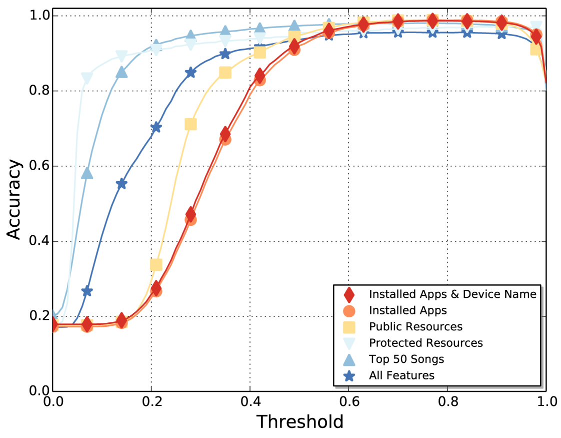
Next, we implement a bag-of-features model and proceed to evaluate a linear SVM-based classifier. In this case, each fingerprint is embedded in a feature vector space where each dimension indicates the presence of a unique feature value in the fingerprint. The original data can therefore be represented as a sparse matrix with very high dimensionality. We split the dataset into 3 chronologically sorted sets for training (60%), validating (20%) and testing (20%). The model which perfoms best on the validation split is used to compute the performance metrics presented in Table 3.

As can be seen, this classifier generally performs quite well and can provide more balanced results in certain cases (highlighted in bold font in Table 3). Specifically, we observed that the SVM classifier gains advantage over the threshold-based approach when the fingerprint data becomes less significant, i.e. when subtle differences in configuration settings and hardware specifics need to be determined. In such cases, where only restricted and less meaningful data is available, more sophisticated learning approaches could improve the overall fingerprinting accuracy.

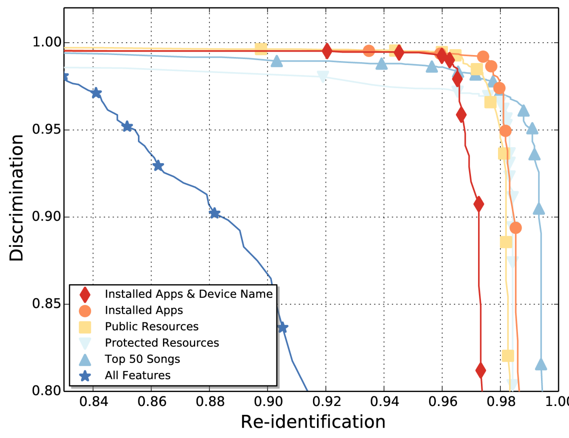
However, the usually high dimensionality of the problem incurs a massive use of resources and considerably increases the time required to train a model. In particular, when all features are considered, the vector space represents more than 3.13×10^6 unique features from our dataset. An Amazon EC2 instance with 244GB of RAM has not been sufficient to complete such an experiment. In a real-life scenario where new fingerprints are collected in a continuous fashion, the problem would soon become intractable with such a classifier. On the contrary, our low complexity method based on an optimized threshold is much faster, requires fewer resources and still achieves a slightly better performance on most feature sets.

6 Discussion

It can be seen that our fingerprinting approach achieves highly accurate results throughout. Whereas hardware identifiers were previously used to track users [3, 4], user configuration profiles now provide sufficient information. In most cases, the total accuracy of our evaluations was way over 90%. In the best-case scenarios, it was almost 98%. This proves that the configuration of today's mobile devices is so highly personalized—the data is so heterogeneous—that it can be used for fingerprinting with a high level of accuracy.



(a) Accuracy vs. Threshold



(b) Discrimination vs. Re-identification

Fig. 3. Evolution of Accuracy during Training and Performance Metrics Trade-Off

Feature Set	Testing Acc.	Discr. Acc.	Re-ident. Acc.
Installed Apps (URL) & Device Name	91.83%	94.46%	85.75%
Installed Apps (Icon Cache)	92.28%	95.13%	85.81%
Public Resources	91.85%	94.53%	85.04%
Protected Resources	90.90%	94.34%	83.11%
Top 50 Songs	89.29%	92.97%	80.29%
All	—	—	—
Device Name & WiFi	**83.53%**	85.90%	78.15%
Public w/o Apps, Name, Songs	**73.60%**	75.57%	69.14%
Public w/o Apps, Name, Songs, WiFi	**65.27%**	65.54%	64.63%

Table 3. Comparison of different feature set combinations for bag-of-features + SVM model.

It was apparent during our analyses that the overall performance decreased as more features were taken into account. We determined, for example, that the list of apps installed had, in itself, very good predictive powers. If we expanded this low-dimensional space, the similarity distribution was modified by the other features and, therefore, the overall performance of our classifier decreased (the curse of dimensionality). In this case, strongly distinguishing features were overlaid by noise from the high-dimensional space. Specifically, performance decreased by features which take the same values in fingerprints from different devices (configuration settings), or which fluctuate strongly on the same device (ISP, public IP etc.). Moreover (and in particular for configuration-based fingerprinting), it is important to use a few, well-performing features in order to optimize the re-identification accuracy. From a fingerprinting perspective, good features are those which are balanced in terms of stability and entropy—according to our results, these would include the installed apps, the top 50 songs, and the device name.

With regard to user privacy, the main issue with this new approach is that, in most cases, users were unaware of the data collection taking place and could not prevent it. We also demonstrated that our approach works even with modified configurations, i.e. when individual features are removed by iOS updates or change over time. An ongoing user privacy issue also involves the fact that a fingerprint can also be linked to the user's real-world identity. This was the case, for example, with the iOS sandbox vulnerabilities we reported to Apple. In principle, these enabled any app to read the user's unique Apple ID—completely unnoticed. Even if the sandbox issues have now been resolved with iOS 8, fingerprints could have been correlated with Apple IDs in advance. This would enable a user to be identified via his or her profile, even in the future.

Our method also functions if devices are restored. Whereas Apple's Advertising Identifiers change after a restore or device replacement, most of the features we use for fingerprinting are restored from backups during the restore process. In this sense, our configuration-based identifier is even stronger than any previous hardware identifiers. As long as a user's personal profile does not change significantly, he or she can continue to be identified for an indeterminate amount of time.

6.1 Countermeasures

Ultimately, this leads us to a discussion of potential countermeasures. The strict iOS security model makes it impossible to restrict or modify the data provided by the iOS API. In order to do this, a jailbreak would be required. This would, however, cause essential security mechanisms to be deactivated, and could even cause privacy threats to increase. The only practical way is, therefore, for the manufacturer (in this case, Apple), to further restrict access to strongly distinguishing features. We therefore suggest the following measures: Apple should remove the sandbox defect which enables third-party apps to obtain a complete list of all the apps. Furthermore, identifying the apps installed using URL schemes should also be prevented. For example, a rate limit could be introduced for calling the *UIApplication* method *canOpenURL*. Furthermore, access to the device name (*name* property in *UIDevice*) should be deprecated. We assume that these measures will not limit apps' functions and usability. In addition, we recommend adding the media library (top 50 songs) to the protected resources and permitting access to them only with user consent. If these measures were implemented, our re-identification accuracy would be reduced to around 44%. If Apple were also to restrict access to the WiFi SSID, re-identification accuracy would be reduced to less than 18% (see Table 1).

It should be noted that this decrease in re-identification accuracy is observed only with the threshold-based classifier. As mentioned above, the SVM classifier was still able to fingerprint devices with reasonable accuracy (∼65%) based on restricted feature sets of less significant data. However, we assume that this is, to a certain extent, caused by the small sample size of our experiment. In real- world scenarios, we expect also a drop in performance for the SVM classifier: If access to meaningful and strong distinguishing features is restricted, fingerprinting will be limited to less significant configuration settings and to certain device specifics, such as the device model. The possible values for these features and their permutations are strongly bounded. While in a smaller dataset these subtle configuration differences can still be observed using advanced learning approaches, we expect these differences to blur and to become barely noticeable as the number of fingerprints increases.

6.2 Limitations

One shortcoming of our experiment is the fact that, for privacy reasons, our evaluations could only be conducted on anonymous data. As previously mentioned, any personally identifiable data was anonymized using our data collection app before being transmitted using hashing. This was done in order to preserve the privacy of our voluntary study participants. As information is lost during this process, however, this approach limited our evaluation in certain ways. This becomes particularly apparent when we flatten the fingerprint structure. In order to measure the similarity of hierarchical structures (such as contact lists) we convert the tree of corresponding hashes into a set of tokenized values. This flattening allows for fine-grained comparison on a value level. Although this approach worked very well, it should be noted that flattening also causes a certain degree of information loss. For instance, if two devices reported contact lists containing "John Smith; Adam Lennon" and "John Lennon; Adam Smith" the tokenization and subsequent comparison would lead to a Jaccard similarity coefficient of 1, even though different contacts were compared. We would like to underline that our reported results were affected only marginally by these limitations, if at all. It can be assumed that, in real-world scenarios where personal raw data is available, the overall fingerprinting accuracy would be even higher—–not least because more comparison opportunities arise for measuring the similarity on a string and character level.

Another drawback of our study is the lack of comparison how, apart from support vector machines, other machine-learning techniques would perform on this problem. We agree that other supervised approaches would also be applicable and that there is not just one way of doing this. We argue, however, that our threshold-based classification is still appropriate, for two main reasons: Firstly, most machine-learning methods require a sufficiently large set of classes to learn from.

As several of our users submitted only one fingerprint, we resorted to a variant of k-nearest neighbors, which is known to perform well in such cases. Secondly, the simplicity of our approach once more underlines the practical relevance, makes our results easily reproducible, and confirms the associated privacy risks.

7 Conclusion

In this paper, we describe a new and effective approach to fingerprinting mobile devices. We demonstrate that mobile device configurations have now become so highly personalized that they can be used to create a unique fingerprint for every user. This can, in turn, be used for user tracking. All the information used to generate the fingerprints can be queried by third-party apps via the iOS SDK.

By comparing the similarity of fingerprints, we are able to match them unequivocally to individual devices, even if individual feature values change over time. Specifically, we use real-world data from almost 8,000 different devices to show that a device's fingerprint changes by an average of no more than 10% over a month. We use a supervised learning approach to solve a multiple class classification problem, which, in the end, allows us to match fingerprints to devices with a total accuracy of over 97%. We determine that a user's apps and a user's music taste could serve as a highly personalized, almost unique fingerprinting feature.

From a privacy point of view, it is particularly alarming that users cannot withdraw from this tracking method and that they are unaware that it is taking place. As long as their personal profiles do not significantly change, they will remain identifiable in the long term via the information in the SDK, even if they replace their devices. During our study, we also discovered several iOS vulnerabilities which, until the release of iOS 8, enabled every app to read the unique Apple ID (email address) of a user. If a fingerprint profile is used to link a user's real-world identity via one of these vulnerabilities, the user will, in future, continue to be identifiable by his or her unique device configuration.

In conclusion, we realize that Apple's iOS has made user tracking by third-party apps considerably more difficult than it is on other mobile platforms. In this respect, Apple's elimination of various hardware identifiers has been particularly positive. The highly heterogeneous configuration data of mobile devices, however, is still sufficient to uniquely identify and track users.

References

[1] Apple, "Worldwide Developer's Conference (WWDC) Keynote 2014." http://devstreaming.apple.com/videos/wwdc/2014/101xx36lr6smzjo/101/101_hd.mov.

[2] AppBrain Stats, "Distribution of free vs. paid Android apps." http://www.appbrain.com/stats/free-and-paid-android-applications.

[3] M. Egele, C. Kruegel, E. Kirda, and G. Vigna, "Pios: Detecting privacy leaks in ios applications.," in *NDSS*, 2011.

[4] A. Kurtz, A. Weinlein, C. Settgast, and F. Freiling, "DiOS: Dynamic Privacy Analysis of iOS Applications," Tech. Rep. CS-2014-03, June 2014.

[5] M. C. Grace, W. Zhou, X. Jiang, and A.-R. Sadeghi, "Unsafe exposure analysis of mobile in-app advertisements," in *Proceedings of the fifth ACM conference on Security and Privacy in Wireless and Mobile Networks*, pp. 101–112, ACM, 2012.

[6] Google, "Android Developer Reference, TelephonyManager, getDeviceId()." http://developer.android.com/reference/android/telephony/TelephonyManager.html#getDeviceId().

[7] Y. Agarwal and M. Hall, "Protectmyprivacy: detecting and mitigating privacy leaks on ios devices using crowdsourcing," in *Proceeding of the 11th annual international conference on Mobile systems, applications, and services*, pp. 97–110, ACM, 2013.

[8] Apple, "Worldwide Developer's Conference (WWDC) Keynote 2014." http://devstreaming.apple.com/videos/wwdc/2014/101xx36lr6smzjo/101/101_hd.mov.

[9] P. Eckersley, "How unique is your web browser?," in *Privacy Enhancing Technologies*, pp. 1–18, Springer, 2010.

[10] N. Nikiforakis, A. Kapravelos, W. Joosen, C. Kruegel, F. Piessens, and G. Vigna, "Cookieless monster: Exploring the ecosystem of web-based device fingerprinting," in *Security and Privacy (SP), 2013 IEEE Symposium on*, pp. 541–555, IEEE, 2013.

[11] G. Acar, C. Eubank, S. Englehardt, M. Juarez, A. Narayanan, and C. Diaz, "The web never forgets: Persistent tracking mechanisms in the wild," in *Proceedings of the 21st ACM Conference on Computer and Communications Security (CCS 2014)*, 2014.

[12] K. Boda, Á. M. Földes, G. G. Gulyás, and S. Imre, "User tracking on the web via cross-browser fingerprinting," in *Information Security Technology for Applications*, pp. 31–46, Springer, 2012.

[13] K. Mowery and H. Shacham, "Pixel perfect: Fingerprinting canvas in html5," *Proceedings of W2SP*, 2012.

[14] J. Lukas, J. Fridrich, and M. Goljan, "Digital camera identification from sensor pattern noise," *Information Forensics and Security, IEEE Transactions on*, vol. 1, no. 2, pp. 205–214, 2006.

[15] M. Chen, J. Fridrich, M. Goljan, and J. Lukás, "Determining image origin and integrity using sensor noise," *Information Forensics and Security, IEEE Transactions on*, vol. 3, no. 1, pp. 74–90, 2008.

[16] O. Çeliktutan, B. Sankur, and I. Avcibas, "Blind identification of source cell-phone model," *Information Forensics and Security, IEEE Transactions on*, vol. 3, no. 3, pp. 553–566, 2008.

[17] J. R. Corripio, A. Sandoval Orozco, L. Garcia Villalba, J. C. Hernandez-Castro, and S. J. Gibson, “Source smartphone identification using sensor pattern noise and wavelet transform,” 2013.
[18] H. Bojinov, Y. Michalevsky, G. Nakibly, and D. Boneh, “Mobile device identification via sensor fingerprinting,” *arXiv preprint arXiv:1408.1416*, 2014.
[19] A. Das, N. Borisov, and M. Caesar, “Do you hear what i hear? fingerprinting smart devices through embedded acoustic components,” 2014.
[20] Z. Zhou, W. Diao, X. Liu, and K. Zhang, “Acoustic fingerprinting revisited: Generate stable device id stealthy with inaudible sound,” *arXiv preprint arXiv:1407.0803*, 2014.
[21] S. Dey, N. Roy, W. Xu, R. R. Choudhury, and S. Nelakuditi, “Accelprint: Imperfections of accelerometers make smartphones trackable,” in *Proceedings of the Network and Distributed System Security Symposium (NDSS)*, 2014.
[22] S. Seneviratne, A. Seneviratne, P. Mohapatra, and A. Mahanti, “Predicting user traits from a snapshot of apps installed on a smartphone,” *ACM SIGMOBILE Mobile Computing and Communications Review*, vol. 18, no. 2, pp. 1–8, 2014.
[23] A. J. Oliner, A. P. Iyer, I. Stoica, E. Lagerspetz, and S. Tarkoma, “Carat: Collaborative Energy Diagnosis for Mobile Devices,” *Proceedings of the 11th ACM Conference on Embedded Networked Sensor Systems*, pp. 10:1—-10:14, 2013.
[24] Apple, “Submitting Apps that Use the Advertising Identifier.” https://developer.apple.com/news/?id=08282014a.
[25] Y. Lechelle, “OpenUDID.” https://github.com/ylechelle/OpenUDID.
[26] Apple, “iOS 7.0 Release Notes.” https://developer.apple.com/library/ios/releasenotes/General/RN-iOSSDK-7.0/.
[27] Y. Lechelle, “OpenIDFA.” https://github.com/ylechelle/OpenIDFA.
[28] Apple, “iOS Developer Library.” https://developer.apple.com/library/ios/navigation/.

A iOS Fingerprinting Features

	Feature	Type	Stability
Public Ressources	Closed Captioning enabled?	Boolean	1.0000
	Guided Access enabled?	Boolean	0.9996
	In-app purchases allowed?	Boolean	0.9960
	Inverted Colors enabled?	Boolean	0.9986
	Language != Country?	Boolean	0.9988
	Mono Audio enabled?	Boolean	0.9984
	Twitter set up?	Boolean	0.9982
	VoiceOver enabled?	Boolean	0.9996
	VoIP allowed?	Boolean	0.9993
	Jailbreak?	Boolean	0.9986
	Carrier Name	String (medium range)	0.9984
	Internet Connection Type	Boolean	0.9417
	Current ISP	String (medium range)	0.8390
	Current Public IP	String (medium range)	0.6984
	Device Country	String (medium range)	0.9989
	Device Language	String (medium range)	0.9984
	Device Model	String (medium range)	0.9995
	Device Name	String (high range)	0.9810
	iOS Version	String (low range)	0.9988
	Installed Apps (Icon Cache)	List (many elements)	0.7527
	Installed Apps (URL Schemes)	List (many elements)	0.9582
	Installed Keyboards	List (few elements)	0.9960
	Top 50 Songs	List (many elements)	0.9202
	WiFi SSID	String (high range)	0.8294
Protected	Calendar Names	List (few elements)	0.8873
	Contacts	List (many elements)	0.8191
	Photo Album Titles	String (high range)	0.8848
	Reminder List Names	String (high range)	0.8921
	Twitter Account Name	String (high range)	0.9598

Table 4. List of iOS fingerprinting features and their stabilities.