Sashank Narain* and Guevara Noubir

Mitigating Location Privacy Attacks on Mobile Devices using Dynamic App Sandboxing

Abstract: We present the design, implementation and evaluation of a system, called MATRIX, developed to protect the privacy of mobile device users from location inference and sensor side-channel attacks. MATRIX gives users control and visibility over location and sensor (e.g., Accelerometers and Gyroscopes) accesses by mobile apps. It implements a *PrivoScope* service that audits all location and sensor accesses by apps on the device and generates real-time notifications and graphs for visualizing these accesses; and a *Synthetic Location* service to enable users to provide obfuscated or synthetic location trajectories or sensor traces to apps they find useful, but do not trust with their private information. The services are designed to be extensible and easy for users, hiding all of the underlying complexity from them. MATRIX also implements a *Location Provider* component that generates realistic privacy-preserving synthetic identities and trajectories for users by incorporating traffic information using historical data from Google Maps Directions API, and accelerations using statistical information from user driving experiments. These mobility patterns are generated by modeling/solving user schedule using a randomized linear program and modeling/solving for user driving behavior using a quadratic program. We extensively evaluated MATRIX using user studies, popular location-driven apps and machine learning techniques, and demonstrate that it is portable to most Android devices globally, is reliable, has low-overhead, and generates synthetic trajectories that are difficult to differentiate from real mobility trajectories by an adversary.

Keywords: Location Privacy Protection, Anonymity, Android Audit Framework, Synthetic Mobility Models, Location-based Services, Mobile Apps, Android

DOI 10.2478/popets-2019-0020
Received 2018-08-31; revised 2018-12-15; accepted 2018-12-16.

***Corresponding Author: Sashank Narain:** College of Computer and Information Science, Northeastern University, Boston, MA, USA, E-mail: sashank@ccs.neu.edu
Guevara Noubir: College of Computer and Information Science, Northeastern University, Boston, MA, USA, E-mail: noubir@ccs.neu.edu

1 Introduction

Modern mobile smartphones are equipped with a large number of precise and sophisticated sensors. These sensors vastly improve the quality of the user's interaction with the environment, but also pose significant threats for privacy breaches by directly or indirectly leaking private information about their users. The leakage of location information from the GPS sensor, for instance, has been a fast growing privacy concern. The commercial GPS hardware available in modern smartphones is capable of triangulating a user's position within an accuracy of 3 meters. This leakage enables more sophisticated threats such as tracking users, identity discovery, and identification of home and work locations.

Motivation: The current protections against location tracking mostly revolve around obfuscating the users' location. Previous work proposed solutions that induce noise in the location data [4, 9, 18, 73, 80]. Others devised solutions that send the real location with dummy locations or within a data-set, and use the response pertaining to the real location [41, 48, 49, 56, 75, 85]. Others proposed stripping off all identifying information about a user before sending the real location data in order to protect the user's privacy [40, 71]. Unfortunately, these solutions still leak information about their users and can be combined with other information (e.g., census data) to infer user identities and their locations [72, 86]. Mobile operating systems also try to prevent undesired location tracking by implementing permissions that all apps must request for accessing location data. These measures, however, are not very effective in preventing location tracking because users are unaware of an app's privacy practices and are often careless about granting such permissions. Also, no protections exist against sensor side-channels (e.g., from Accelerometers, Gyroscopes, and Magnetometer) even when they are now known to leak location information [17, 37, 59, 60, 62, 64], enable unauthorized keyboard logging [63], and covert channels [16]. Localizing a user is important beyond privacy breaches due to tracking, it can be used to trigger more intrusive attacks such as targeted stealthy man-in-the-middle attacks in some wireless protocols [20].

An alternative protection against location tracking is the generation of synthetic location trajectories [15, 58] that are independent of users real locations [23, 51]. These trajectories guarantee location privacy because it is not possible to derive the user's location from them, however, they risk denial of service if an adversary detects that the trajectories are fake. To be effective against detection, these trajectories must emulate real movements and routes by incorporating real user transitions, movement schedules, traffic information and driving behavior. Synthetic, yet realistic, mobility trajectories are important as they have the potential to eliminate privacy leaks and also enable the understanding of how users' location information is exploited by mobile apps.

Approach: The proposed MATRIX system is designed to address privacy protection weaknesses in Android. To detect leakage from location and sensor data, it implements a PrivoScope service to monitor and analyze apps patterns for accessing location and sensor APIs. PrivoScope provides users with real-time notifications and a graphical interface to display how apps access their location information and permissionless sensors (e.g., the time of location access, the accuracy of the location data, the rate a sensor was sampled, and whether the app was in foreground or background). This user interface helps users make more privacy informed decisions about providing synthetic data to apps using MATRIX or uninstalling/disabling apps they do not trust. PrivoScope also implements a permission-protected API that allows security apps installed on the device to get real-time information about other apps accessing private location and sensors information.

To protect against leakage of location data, MATRIX implements a Synthetic Location Service that gives users the capability of setting their privacy preferences for each installed app. The service dynamically and seamlessly sandboxes apps installed on the device to receive obfuscated or synthetic location and sensor feeds as specified by the user. The synthetic feeds are generated such that they are difficult to distinguish from real ones by an adversary. To this end, we model user identities and their movements between locations through Finite State Machines (FSM) with probabilistic transitions connecting states. The transitions between states represent routes that are generated from graphs constructed from real road networks. These synthetic routes are made realistic by generating a randomized schedule (path in the FSM) using Linear Programming that satisfies each state's preferences in terms of time spent, and expected arrival in those states. We further incorporate traffic information from historical traffic APIs such as Google Maps Directions API, generate accelerations and speeds using Quadratic Programming based on statistical information from user driving behavior, and also add noise to the synthetic data to emulate real data, in addition to incorporating walk times and idle times.

We extensively evaluated MATRIX to validate system performance and reliability, and realism of the synthetic trajectories. Testing with 1000 popular Android apps, we report negligible impact in performance and reliability. For 10 popular location-driven apps, we report that MATRIX is undetected while at least one app could detect non-MATRIX mobility patterns. Our user study involving 100 users indicates that the synthetic trajectories are difficult to differentiate from real traces visually, with more users confusing synthetic trajectories to be real. Our machine learning evaluation indicates that most well-known algorithms fail to differentiate between real and synthetic trajectories with an average accuracy of 50% (comparable to an algorithm that uses a coin-flip), with just one algorithm achieving an accuracy of 63% in guessing if a trajectory is synthetic.

Contributions: Our contributions are as follows:

- MATRIX implements an efficient, reliable and extensible auditing system for the Android ecosystem. It audits all location and sensor accesses by all apps on the device to detect privacy leakages, generates real-time notifications and graphs for visualizing these accesses in an easy and intuitive manner. It can be used by users, security apps and researchers to identify which apps misuse/leak private location and sensors information, by analyzing an app's accesses and injecting synthetic honey-data to observe if it is used in contexts not authorized by users.
- MATRIX gives users the capability to change their privacy preferences, and provide obfuscated or synthetic location and sensor (Accelerometer, Gyroscope and Magnetometer) trajectories to installed apps. We show that generating realistic synthetic location and sensor trajectories is feasible by incorporating traffic information, using randomized yet realistic schedule constraints using a linear program, and matching statistical characteristics of user driving behavior using a quadratic program.
- MATRIX is an extensible and lightweight system integrated within Android without modifications to the operating system, nor requires rooting the device. It is implemented following the Android design paradigms and can be easily extended for other

sensitive APIs, e.g., Wi-Fi, Telephony, Camera and Microphones. MATRIX attempts to hide all underlying system complexity from the users to provide users an intuitive and holistic view of accesses to their private information, to enable these users to make privacy-aware decisions regarding apps installed on their device.

2 Location Privacy in Android

We discuss the Android location and sensor APIs, Android privacy protection schemes and their weaknesses.

2.1 Android Location & Sensor APIs

The MATRIX system audits all location and sensor accesses and updates this information reported to an app in some contexts. There are a standard set of Android APIs that provide this information.

Location information can be accessed using four different APIs. The `LocationManager` is the default API available in all versions of the Android SDK. The `FusedLocationProviderClient`, `FusedLocationProviderApi` (deprecated) and `LocationClient` (deprecated) are provided by Google Play services as recommended closed source alternatives that consume less battery for higher accuracy data. All these APIs contain `request*` and `remove*` calls (e.g., `requestLocationUpdates` in `LocationManager`) that enable apps to register and unregister for continuous location updates. Once registered, location information is sent asynchronously to the listeners based on the criteria set by the app (e.g., quality, rate, latency). These managers also contain additional methods such as `getLastKnownLocation` in `LocationManager` that can return a location update immediately.

Sensor information (e.g., Accelerometers and Gyroscopes) can be accessed using the `SensorManager` API. It is important to note that access to these sensors does not require permissions in any versions of Android. Also, these sensors can be accessed by apps without any notification or visual cues to the user.

2.2 Weaknesses in Privacy Protections

The location privacy protection schemes implemented by Android are not sufficient for completely protecting a user's privacy. Some of the weaknesses are discussed below. Note that these weaknesses are labeled (**W#**) for ease of referring to them in the next section.

Weak Permissions Model (W1): Android specifies two permissions for limiting access to the user's location information: `ACCESS_FINE_LOCATION` and `ACCESS_COARSE_LOCATION`. The former allows apps to access high accuracy location information, while the latter provides obfuscated information to hide the user's real location. The permissions model is a good step in notifying users of location access, however, this protection is limited as users have an option to always allow access. This means that the user will not be notified about location access again even if the app's context has changed, i.e., location is accessed from another activity or from a service, or a previously benign app is updated with a privacy intrusive version. Moreover, the obfuscated locations still leak information about the real locations. There is currently no mechanism for users to completely hide their location by providing synthetic information to untrusted apps.

Non-existent Auditing Capabilities (W2): Android does not provide a framework to audit how apps access a user's private information. Also, App stores (e.g., Google Play Store) do not provide enough information about the privacy practices of an app. Without any privacy-related knowledge, users are more than likely to install and use an app if they require the services provided by that app.

Weak Location Activity Notification (W3): The Android operating system displays a notification icon on the notification bar of the device, whenever any app requests continuous location updates. An adversary can easily bypass this notification icon by using an alternative method for location access. One example is the `getLastKnownLocation` call in `LocationManager` which can be invoked numerous times for receiving continuous location updates. Another example is exploiting the permissionless sensors like Accelerometers, Gyroscopes and Barometers to infer user locations. In any case, the notification simply indicates that some app has access to location and no further information is given to the user to make privacy-aware decisions.

Restricted Privacy Preferences (W4): Android does not provide the capability for users to define their privacy preferences for apps installed on their device. Users can deny location access to certain apps by disallowing location permissions, however, certain apps may then deny service to the users. There are situations in which users may not wish to disclose their locations, in particular at some moments in time, and still require

the services of the app. One example of this is when the app is turned-off or in the background.

3 High-Level Approach

MATRIX is an extensible system designed to address the above location privacy protection weaknesses in Android. It is implemented using Android design paradigms for easy integration into the Android ecosystem with minimal changes. It is meant to be easy to use and intuitive for end-users. The system comprises of an App-activity PrivoScope Service, a Synthetic Location Service, and a Synthetic Location Provider.

The App-activity PrivoScope Service monitors and analyzes apps patterns of location and sensor API accesses. It is designed for end-users, security apps and researchers desiring to assess the privacy posture of installed apps on the device. End-users can view all location and sensor access information as intuitive graphs. Other apps can get real-time audit events via a permission protected secure API **(W2)**. The service also displays real-time visual notifications of location and sensor access activity to users. The notification bar is updated whenever any app accesses these sensors and displays information about which apps are actively accessing what sensors on the device **(W3)**. The architecture of PrivoScope is described in Section 4.2.

The Synthetic Location Service provides a user interface for setting the location and sensor privacy preferences for all installed apps. This service implements three privacy settings: Block level, City level and Synthetic Locations **(W4)**. It relies on the default Android permission manager for managing location permissions, however, restricts location access to background apps by default. Instead of completely denying location information, the service detects if the app is in the background and provides it the last location fix that the app received in foreground to prevent it from tracking users **(W1)**. The architecture of this service is described in Section 4.3. The Synthetic Location Provider provides the Synthetic Location Service obfuscated/synthetic locations whenever the service requests for it **(W1)**. The techniques for modeling and generating synthetic identities and movements are described in Section 5.

Figure 1 shows how MATRIX integrates into the Android ecosystem. The integration is closely aligned with current Android design paradigms with an assumption that the paradigm will not change significantly as Android evolves in the future. The PrivoScope Service

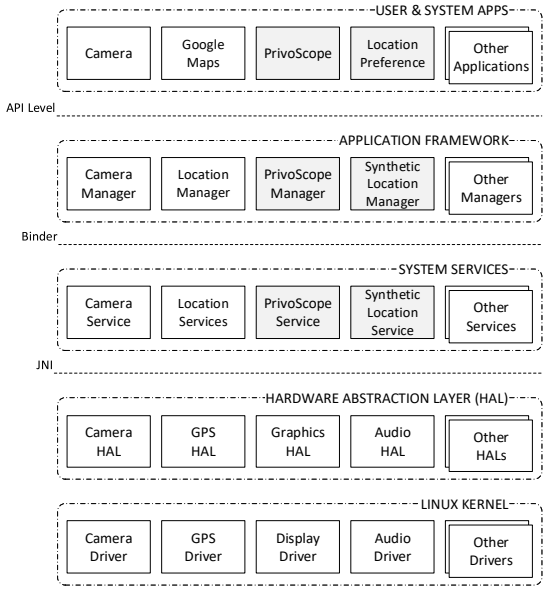


Fig. 1. MATRIX integration into the Android ecosystem.

and Synthetic Location Service are implemented as system services that start at device boot and are registered in the system server registry. These services implement all the protections that ensure that only authorized apps can use their functions. Apps installed on the device interact with these services using APIs provided by the PrivoScope Manager and Synthetic Location Manager. These managers are loaded into each app's process and communicate with the corresponding services. At the user level, MATRIX implements a PrivoScope GUI that provides a graphical interface to the users to analyze the app's privacy practices, and a Location Preference GUI that enables users to set their location privacy preferences. These also use the PrivoScope and Synthetic Location Managers to communicate with the corresponding system services.

4 MATRIX Architecture

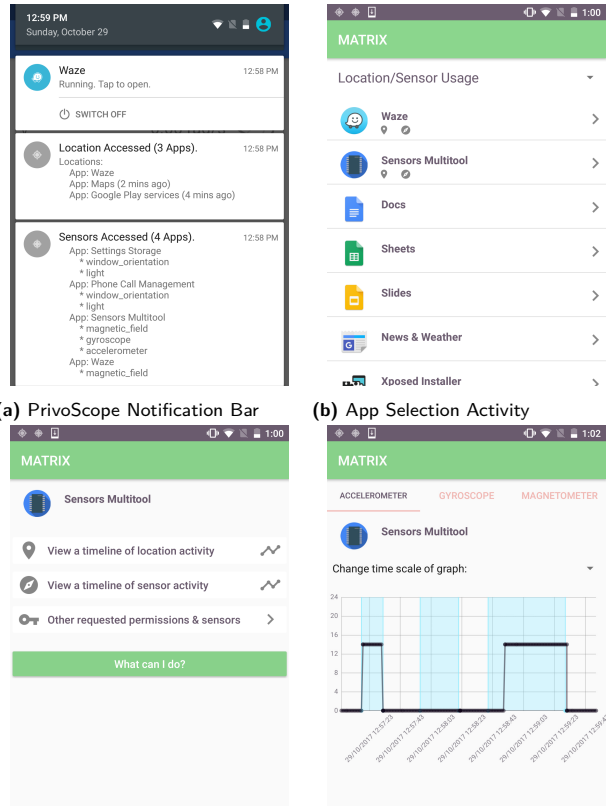
This section describes the architecture of the PrivoScope and Synthetic Location services implemented for the MATRIX system.

4.1 API Call Interception

Previous mitigation systems (excluding Boxify [12]) were implemented by either modifying the Android source code, using rooted devices, or using third party frameworks such as the Xposed Framework [84]. The

Xposed framework adds an extended `app_process` executable in the `/system/bin` folder of the device on installation. This extended `app_process` adds an additional `jar` file to the classpath and calls methods even before the `main` method of `Zygote` is called. This enables apps to intercept method calls that are otherwise inaccessible from an app's process.

MATRIX uses the Xposed framework to intercept location and sensor API calls. One example usage in our context is intercepting the `requestLocationUpdates` method of `LocationManager` to generate an event every time an app requests location updates. This event contains all the relevant information about the request, which is sent to PrivoScope for logging and notification. Using the framework is both necessary and advantageous due to the following reasons: (1) Xposed has the capability to intercept external APIs like Google Play Services which is currently not possible by modifying the Android source or by rooting, (2) the framework is supported and has a consistent API for different versions of Android ensuring portability and ease of development, and (3) the framework does not require a rooted device to function properly. We developed a simple tool that automates the installation of Xposed and MATRIX through a custom recovery (e.g., TWRP [77]) without rooting the device. The Xposed framework and TWRP recovery are both open-source and consistently analyzed and updated by a large community of Android users, making them quite reliable.



(a) PrivoScope Notification Bar **(b)** App Selection Activity

(c) App Detail Activity **(d)** Accelerometer Timeline

Fig. 2. Example screenshots of the PrivoScope GUI.

4.2 The App-activity PrivoScope Service

The goal of the PrivoScope service is to give users an intuitive interface to help them make privacy aware decisions regarding installed apps. We assume that the user has some knowledge of privacy leakages and are comfortable using their devices (e.g., checking notifications, opening apps). To achieve this goal, the service hides all the underlying system complexity from the users. This paper focuses on the high level design of the services as outlining all implementation challenges is outside scope due to length requirements. We mention just a few here to provide an idea of the system complexity: (1) the services must start before all user services mandating that the services be registered in the system server registry without modifications to the Operating System, or rooting the device, (2) these services run within a privileged environment and interfaces must be defined using `Binder` for other apps to communicate with them, (3) by design, the interfaces allow all apps to communicate with the services mandating security checks to be implemented within the services, and (4) these security checks and service functionality must be very lightweight and efficient to not cause device instability and process multiple requests quickly.

At a high level, the PrivoScope service uses the Xposed framework to intercept all location and sensor APIs, generates events containing the audit details, adds the events to a database and displays real-time usage notifications to the end-user. The service also exposes a permission protected API that other security apps can register to get real-time and archived audit events. Figure 2 shows example screenshots of the PrivoScope GUI, where Figure 2a shows the PrivoScope real-time location and sensor usage notification, Figure 2b shows a list of installed apps sorted by most recent access of location and sensor APIs, Figure 2c shows links to an app's permissions and access details, and Figure 2d shows a timeline of Accelerometer access by an app at different times. This timeline can be set to display accesses in the past month, week, day, or a custom number of hours. Note that an app's life-cycle is color coded to help users differentiate between foreground and background accesses. Here, blue indicates that the app

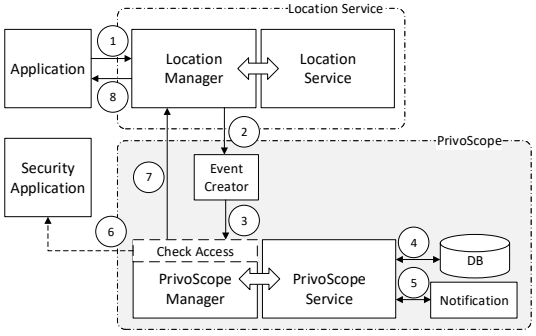


Fig. 3. Architecture diagram of the PrivoScope service.

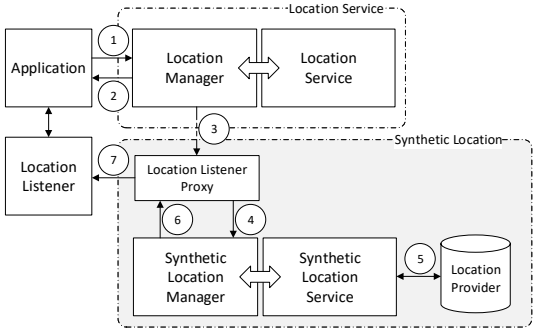


Fig. 4. Architecture diagram of the Synthetic Location service.

was in the foreground while gray would indicate background access. The performance analysis and evaluation of PrivoScope are reported in Sections 6.1 and 6.2.

The architecture of the PrivoScope service for a `requestLocationUpdate` method call from `LocationManager` is shown in Figure 3. Note that this architecture is generic across all location managers and the sensor manager and we use `LocationManager` here just for illustration purposes. Like all other Android services, PrivoScope implements a manager called `PrivoScopeManager` that exposes public APIs to other apps and a service called `PrivoScopeService` that performs all the security sensitive operations and checks if apps have appropriate access rights for its services.

The control flows like this: An app requests continuous location updates using the `requestLocationUpdate` method call from `LocationManager`. The manager and the privileged `LocationManagerService` validate the app's access by checking its requested permissions ①. Once access is validated, the API call interception service generates an event containing all relevant information to be logged for auditing. All private user information contained by the request are ignored. For example, this specific event would contain *the system time, the app package name, the activity invoking the request, whether the app is background or foreground, the requested location provider, and the requested accuracy and sampling rate* ②. This event is then sent to the `PrivoScopeManager` for logging using an `addAuditEvent` method call exposed by the manager ③. The `PrivoScopeManager` forwards this event to the `PrivoScopeService` which validates whether the package name in the event is the same as the package name of the app making the request. This ensures security as only apps generating an event can add the event. The event is discarded if the package names do not match and a `SecurityException` is thrown. In case of a successful match, the event is added to the service's database ④. The `PrivoScopeService` also sends this event to a Notification service that keeps track of all active apps accessing location and sensor APIs and updates the notification bar with this new event information ⑤. The `PrivoScopeManager` exposes a `requestAuditEvents` method call that other apps on the device can register for receiving real-time audit events. This call is protected using a custom permission called `GET_AUDIT_EVENTS` and apps must request this permission for access. The `PrivoScopeManager` sends the event to all registered apps that receive this event asynchronously using a `AuditEventListener` callback interface ⑥. Based on whether this event was successfully added to the database or not, the `addAuditEvent` method call returns a boolean value to the `LocationManager` ⑦. Note that steps ③ to ⑦ execute in a new thread to ensure that the app functionality and the performance is not impacted by PrivoScope. After step ③, the `requestLocationUpdate` method call simply terminates as its return type is a `void`. The other method calls and managers return the expected values and their functionality is not updated by PrivoScope ⑧.

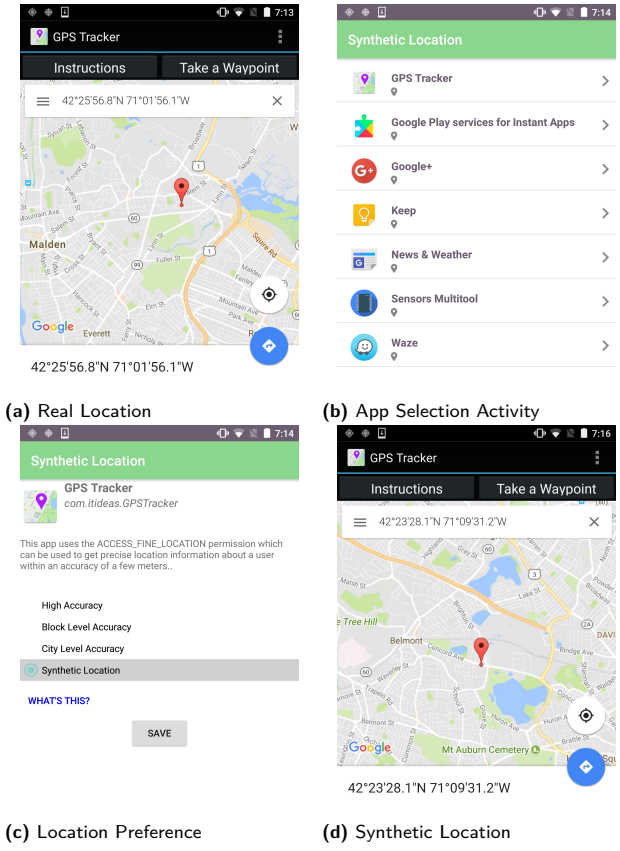
4.3 The Synthetic Location Service

The architecture of the Synthetic Location service is shown in Figure 4, again in the context of receiving location updates from the `LocationManager` API. Like PrivoScope, this architecture is generic across all the location managers. The Synthetic Location service implements a manager called `SyntheticLocationManager` that exposes public APIs to other apps and a service called `SyntheticLocationService` that manages and protects the database storing the user privacy prefer-

ences, and connects with the LocationProvider to request obfuscated/synthetic locations and sensor feeds.

The control flows like this: When an app requests continuous location updates (with the correct permissions) using the requestLocationUpdates call from LocationManager, the first steps that occur are the listener registration (cf. Section 2.1) and addition of the audit event to the PrivoScope service's database (cf. Section 4.2). ①, ②. After registration is completed, all the location fixes generated by the LocationManagerService are typically sent asynchronously to the app's LocationListener, PendingIntent or LocationCallback implementation. In MATRIX, these location fixes are intercepted by a LocationListenerProxy that proxies it to the app's listener. The proxy works by hooking the Location object that is used by all the managers to send location fixes to the app's listener. This enables it to modify the location object before the app loads the information using the get* method calls (e.g., getLatitude() and getLongitude()) ③. The LocationListenerProxy requests the SyntheticLocationManager to provide an updated location for the app, based on the app's location preference set by the user. The manager forwards this request to the SyntheticLocationService that maintains and protects the database storing the user location preference for each app ④. The SyntheticLocationService looks up the user's location preferences in the database, and communicates with the LocationProvider to request an obfuscated/synthetic location if the user has chosen to receive such location information for the app. The default preference set for an app requesting fine location is block level obfuscated data ($200m$) ⑤. An updated location object is returned to the SyntheticLocationService which forwards it to the SyntheticLocationManager. The SyntheticLocationManager sends this location to the LocationListenerProxy that updates it before the app accesses the location ⑥, ⑦.

The Synthetic Location service currently provides four settings for per-app privacy: Default Accuracy, Block Level Accuracy, City Level Accuracy, and Synthetic Locations. Note that the Default Accuracy and Block Level Accuracy options are only available for apps requesting fine location using the ACCESS_FINE_LOCATION permission. This is because apps that use ACCESS_COARSE_LOCATION permissions already receive coarser location data than that provided by the two options. For block level and city level accuracy, we extended the default Android LocationFudger implementation to support different grid resolutions. The implementation is in com.android.server.location.LocationFudger under the Android source tree [6]. We analyzed this code to find that the real location information is obfuscated in two steps. First, a small random offset value is applied to the location to mitigate against accurate detection of grid transitions when a user crosses a grid boundary. This offset is changed just once every hour to mitigate against location inference attacks. Second, the primary means of obfuscation is to snap the above value (already mitigated against grid transitions) to a grid. This grid radius chosen by most recent versions of Android is $2000m$. Note that a thorough security analysis of this technique is outside the scope of this paper, however, this technique has been implemented in all Android devices since 2012 with no known attacks till date. The current grid radius settings for block level and city level accuracy are $200m$ and $5000m$, respectively.



(a) Real Location **(b)** App Selection Activity

(c) Location Preference **(d)** Synthetic Location

Fig. 5. Example screenshots of the Synthetic Location GUI.

Figure 5 shows screenshots, illustrating the Synthetic Location service for a GPS tracking app. Note that this app is used for demonstrating how the service works because it displays the user location on the screen, and it is not a malicious app. Figure 5a shows the test

app displaying the user's real location, Figure 5b shows the list of installed apps that request location permissions, Figure 5c shows the location privacy preference for the test app being changed to synthetic, and Figure 5d shows the test app now displaying a synthetic location in another city. The synthetic locations and corresponding sensor trajectories are generated once a day for every user, and the location and sensor feeds are provided to the service based on the time of the day.

5 Generating Synthetic Identities

We define a synthetic identity as a virtual identity of a smartphone user that does not retain any location specific attributes of the user. MATRIX generates a unique synthetic identity for every user using the system, each with their own unique movement patterns. The identities are generated such that they emulate real user movements, yet, do not contain a real person's movement patterns. Naturally, such movements must adhere to real home and work places, realistic schedules, real driving patterns incorporating both current traffic and user driving behavior. We must emphasize that this synthetic identity is only applied to apps selected by the smartphone user, and other installed apps receive the real location (obfuscated upto $200m$ by extending the Android LocationFudger implementation). This is to ensure that location-based apps can still provide their services with sufficient accuracy without collecting precise location information. The user always has the option to change their privacy preferences and can set them to high accuracy for apps requiring precise location, such as Google Maps. While the synthetic data is consistent with itself (e.g., driving between a synthetic identity home to a synthetic identity work place), it is independent of the real users patterns. Therefore, one current limitation of the trajectory generation scheme is the absence of a method to synthesize movements that preserve a user's driving profile such as distances and speeds. We discuss the limitations of our approach and possible future research directions in Section 5.3. Next, we provide a detailed description of our technique for generating unique and realistic synthetic identities and mobility trajectories for each user.

5.1 Modeling User States

A user's synthetic mobility patterns are defined as an automated probabilistic state machine with a finite set of S states $Q = \{Q_0, \ldots, Q_{S-1}\}$. The states, in this context, represent a set of tuples $\{(\mathrm{Loc}(Q_i), t_{min,i}, a_{min,i}, a_{max,i})\}$, where $\mathrm{Loc}(Q_i)$ is the geographic coordinates of state Q_i, $t_{min,i}$ is the minimum time spent in the state, and $a_{min,i}$, $a_{max,i}$ are the lower and upper time bounds for arrival at the state. The geographic coordinates of the states are obtained from OpenStreetMap by parsing the 'building' and 'amenity' tags [67, 68] of all ways and nodes for the given area. For instance, a 'Home' state can be chosen as a way or node in OpenStreetMap whose building type is one of the following: 'apartments', 'house', 'residential', or 'bungalow'. Similarly, a 'Work' state can be chosen from the 'commercial' or 'industrial' tags. The other attributes are used for scheduling the user's activity for each day and set based on typical times that these activities occur. Note that the attributes are set to default values when they are unimportant for a state, i.e., $t_{min,i} = 0$, $a_{min,i}$ = 00:00:00, and $a_{max,i}$ = 23:59:59. In the simplest form, a state machine may contain just two synthetic states $Q = \{Q_0, Q_1\}$, where Q_0 = 'Home' and Q_1 = 'Work'. We label these as *significant* states as the user spends most of their time in one of these states. The geographic coordinates $\mathrm{Loc}(Q_0)$ and $\mathrm{Loc}(Q_1)$ are randomly chosen from the list of all locations with the relevant tags. Assuming no 'Work from Home' scenarios, the probabilities $P(Q_0)$ and $P(Q_1)$ of occurrence of these states is taken to be 1.

The state machine is made more realistic by adding synthetic states like Q_2 = 'School', Q_3 = 'Gas Station', Q_4 = 'Lunch' and Q_5 = 'Dinner'. We label these as *transitional* states because a user will temporarily visit these states when transitioning between *significant* states (i.e., Q_0 and Q_1). For any transitional state Q_i, the geographic coordinates $\mathrm{Loc}(Q_i)$ is selected from a set of locations $\mathrm{Loc} = \{Loc_1, \ldots, Loc_N\}$ with the relevant tags, such that its euclidean distance is shortest from the significant states, i.e., $\mathrm{Loc}(Q_i) = \arg\min_{L \in \mathrm{Loc}} d(L, \mathrm{Loc}(Q_0)) + d(L, \mathrm{Loc}(Q_1))$. Note that, unlike *significant* states, visits to *transitional* states are occasional based on some specific frequency of occurrence. This frequency, denoted by f_i, is derived from a uniform distribution $\mathcal{U}(l, u)$ with l and u as the bounds for the frequency of visits to that state (e.g., once a week to once a month). In case of 'Gas Station' specifically, the system chooses a random mileage m and gas capacity c, and calculates the frequency as the number of days a user can travel between the *significant* states before the gas level goes below $1/4^{th}$ of capacity, i.e., $f_3 = int(\frac{0.75mc}{d(\mathrm{Loc}(Q_0),\mathrm{Loc}(Q_1)) + d(\mathrm{Loc}(Q_1),\mathrm{Loc}(Q_0))})$. Assuming W workdays in a year, the probability of occurrence for any *transitional* state Q_i is then calculated

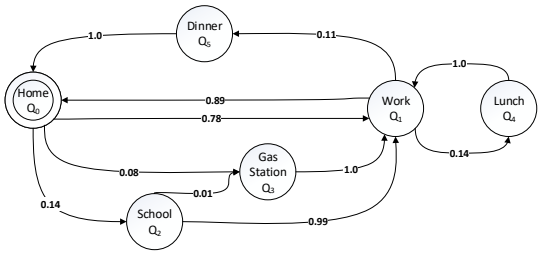


Fig. 6. Example of a simplified finite state machine simulating a user's movements based on some transition probabilities.

as $P(Q_i) = (W/f_i)/W$. Note that the user state timing constraints and above frequencies can be customized for each region using the demographic information available for that region (e.g., USA Census Bureau [66]).

The transition probability between states Q_i and Q_j, denoted by $\chi_{i,j}$, is equivalent to the compound probability of the two independent states, i.e., $P(\chi_{i,j}) = P(Q_i)P(Q_j)$. The following conditions determine if a state Q_i can transition to state Q_j: (1) Q_i is a *significant* state and the originating state for Q_j, (2) Q_j is a *significant* state and the destination state for Q_i, or (3) the two states originate from the same *significant* state Q_s and the route distance $d(\text{Loc}(Q_s), \text{Loc}(Q_i)) < d(\text{Loc}(Q_s), \text{Loc}(Q_j))$. The *significant* states are always connected and their probabilities are calculated as $P(\chi_{0,1}) = 1 - \sum_{i=2}^{S-1} P(\chi_{0,i})$ and $P(\chi_{1,0}) = 1 - \sum_{i=2}^{S-1} P(\chi_{1,i})$, respectively. All other transitions have a probability of 0.

Note that users can go for 'Lunch' in the afternoon and 'Dinner' in the evening from the 'Work' state. If we use the same 'Work' state for both transitions, the probabilities are split when they clearly are different transitions. To address this, the 'Work' state is internally represented as two states: Q_{1a} for afternoon and Q_{1e} for evening. Also note that the model described here is for weekdays, and a similar model is created for weekends with a different set of states (e.g., the user may leave from 'Home' to watch a 'Movie', eat 'Dinner' and return 'Home'). The model can also be easily extended to incorporate multiple similar states such as going to different restaurants for 'Dinner'.

Figure 6 provides an intuition for our automated finite state machine model. This specific model comprises of 6 states $Q = \{Q_0, \cdots, Q_5\}$ and their transition probabilities are shown. We see that it is possible to transition from state Q_0 to states Q_1, Q_2 or Q_3. As the transition probability $P(\chi_{0,1})$ is 0.78, the model should typically choose state $Q_1 \approx 8$ times out of 10. This makes sense as a user will mostly go to 'Work' from 'Home' but may sometimes need to drop their kids to 'School' or fill up gas at a 'Gas Station'.

5.2 Modeling Mobility Trajectories

The finite state machine generated for a user is used to synthesize mobility trajectories for that user every day. This is a 4 step process: (1) synthesize the user states for the entire day, (2) synthesize the schedule to satisfy the time constraints, (3) synthesize the GPS trajectory based on the schedule, and (4) synthesize the sensor trajectory using the GPS trajectory.

Synthesizing the user states: The state machine of a user is loaded every day to generate a route of the states the user will visit that day. This route always starts and ends at the initial state Q_0 ('Home') and traverses through Q_1 ('Work'), i.e., $R = [Q_0, \ldots, Q_1, \ldots, Q_0]$. The first state Q_0 can transition to any connected state Q_i based on the transition probabilities of Q_0. The state Q_i can then transition to any of its connected state Q_j based on the transition probabilities of Q_i, and so forth forming a chain that ends at the final state Q_0. Note that the construction technique of the state machine ensures that this route traverses through Q_1. Let $P(\chi_i) = \{P(\chi_{i,0}), \ldots, P(\chi_{i,S-1})\}$ denote the set of all transitional probabilities of state Q_i. To obtain the next state, the system first derives a random transitional probability from a uniform distribution $\mathcal{P} = \mathcal{U}(0,1)$. This probability $\mathcal{P}$ is then compared with the cumulative probabilities of all transitions in $P(\chi_i)$. A state Q_j is selected if $\mathcal{P}$ lies between the previous state's cumulative probability and its cumulative probability, i.e., $P(X \leq \chi_{i,j-1}) < \mathcal{P} \leq P(X \leq \chi_{i,j})$.

Synthesizing the schedule: A realistic schedule should satisfy the time constraints set for every state in a user's state machine, such as arriving at work between $8am$ and $9am$ or dropping children to school before $8{:}30am$. The schedule should also satisfy the amount of time spent in each state, such as working for at least $8hrs$. The schedule should also account for the time spent in transitioning from one state to the next, such as driving for $0.5hrs$ to get from home to work. All these constraints can be formulated as linear equalities or inequalities, therefore, defining the problem of scheduling as a Linear Program (LP). Let t_i^a and t_i^d be the arrival and departure times at/from state Q_i. The above constraints can be formulated as follows: arriving at state Q_i between $8am$ and $9am$ is formulated as $8am < t_i^a \leq 9am$, specifying that the user works at

least $8hrs$ is formulated as $t^d_{i+1} - t^a_i \geq 8.0$, and the time spent in transitioning from home to work is formulated as $t^a_{i+1} - t^d_i = 0.5$. Naturally, all the times are specified in UTC for consistency and bounded by the day's limits (i.e., 00:00:00 - 23:59:59).

This set of linear equality and inequality constraints define a *convex polytope* of all the schedules satisfying the state constraints, and the transition time constraints between the states. Let $T = (t^a_1, t^d_1, \ldots, t^a_S, t^d_S)$ denote a vector of all the arrival and departure time instants for a route containing S states. One simple way of finding a point on this polytope is by defining an objective function for the vector T with random coefficients, i.e., $c = (c_1, \ldots, c_S)$ where $c_i \in [-1, 1]$. The LP is formally defined as

$$\begin{aligned}
&\text{Maximize} && \sum_{i=1}^{S}(c_i t^a_i + c_i t^d_i) && \text{where } c_i \in [-1,1]\\
&\text{Subject to:} && a_{min,j} < t^a_j \leq a_{max,j} && \text{for } j = 1,2,\ldots,S\\
&&& t^d_{j+1} - t^a_j \geq t_{min,j} && \text{for } j = 1,2,\ldots,S-1\\
&&& t^a_{j+1} - t^d_j = t(\chi_{j,j+1}) && \text{for } j = 1,2,\ldots,S-1
\end{aligned}$$

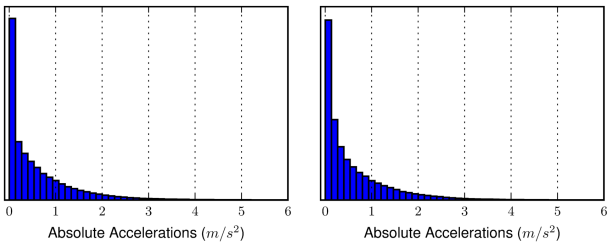
where $t(\chi_{i,j})$ denotes the total time spent in transitioning between two states Q_i and Q_j, $t_{min,i}$ specifies the minimum time spent in state Q_i and $a_{min,i}$, $a_{max,i}$ specify the time bounds of arrival at the state Q_i (cf. Section 5.1).

Solving this LP identifies a corner of the polytope but not a random element within it. If the coefficients of the objective function were repeated, the LP will output the same schedule. To address this, we compute a random point within the polytope by finding different corners of the polytope using random coefficients and then computing a random linear combination of these corners. More precisely, let $C = \{C_1, \ldots, C_N\}$ denote a set of N corners of the polytope obtained using random coefficients, and let $r = \{r_1, \ldots, r_N\}$ denote a set of positive random numbers such that $\sum_{i=1}^{N} r_i = 1$. The random solution defining the user's schedule for that day is then calculated as $Schedule = \sum_{i=1}^{N} r_i C_i$. We observe that $N = 3$ provides a sufficiently random solution and computes the user schedule within 1 second.

Note that as synthesizing the schedule using LP requires pre-calculated transition times $t(\chi_{i,j})$, the system calculates this time using the 'pessimistic' traffic model of *Google Maps Directions API*. The departure time is chosen as the mean of the time constraints for the start state. This typically gives us a worst case transition time between two states and can be used for scheduling. Note that for synthesizing the final trajectory, the 'best_guess' traffic model is used which provides more accurate traffic representation.

Synthesizing the GPS route between two states: The route between two synthetic states is generated using a graph $G = (V, E)$ constructed for the area. The system uses the *Dijkstra's* algorithm to find the fastest route between the states, using the length and speed limit information present in each vertex. The resulting route is split into multiple waypoints based on turns and stop signs (extracted from OpenStreetMap). The source, waypoints and destination are given as input to the *Google Maps Directions API* to obtain historical traffic information about the route. The departure time is specified based on the schedule generated for that day. The route obtained from the Google API consists of multiple steps and can be represented as $R = [r_1, \ldots, r_S]$, where S denotes the number of steps. Each step r_i is attributed with geographic and traffic related information $r_i = (\mathcal{B}, d_{step}, t_{step})_i$, where $\mathcal{B}$ is the list of geographic coordinates of this step, d_{step} is the length of this step in meters, and t_{step} is the time to traverse this step in seconds.

To generate realistic trajectories, all steps of a route must incorporate user driving behavior while also adhering to the step's traffic constraints, i.e., d_{step} and t_{step}. To understand and calculate statistical attributes for user driving behavior, we analyzed 400 driving routes collected from 2 drivers and 4 phones (LG Nexus 5, LG Nexus 5X, Samsung Note 4, and Google Pixel). These routes covered a distance of $\approx 1400kms$ in a major city of USA consisting of both highway and internal roads, as well as peak and off-peak hours. The acceleration and speed information were extracted from these routes for every second to analyze their distribution. We must emphasize that this small data-set is simply a stand-in for a larger data-set collected from many users. We found the speeds to be randomly distributed, however, the absolute values of accelerations approximate to an exponential distribution (mean $\mu = 0.61$, median $M = 0.34$, and standard deviation $\sigma = 0.79$) shown in Figure 7a. Note that the distribution is an approximation and not truly exponential because $\mu < \sigma$, where $\mu = \sigma$ is a property of exponential distributions. Analyzing individual routes, the range of means of absolute accelerations, denoted by $[|\bar{a}|_{min}, |\bar{a}|_{max}]$, varied between $0.1m/s^2$ and $1.1m/s^2$. The range of standard deviations of absolute accelerations, denoted by $[\sigma(|a|)_{min}, \sigma(|a|)_{max}]$, were between $0.4m/s^2$ and $1.1m/s^2$. The bounds of all acceleration values, denoted by $[a_{min}, a_{max}]$, were between $-7m/s^2$ and $7m/s^2$. The means of the accelerations



(a) Real Accelerations **(b)** Synthetic Accelerations

Fig. 7. Distribution of the absolute values of accelerations for both Real ($\mu = 0.61$, $M = 0.34$, $\sigma = 0.79$) and Synthetic ($\mu = 0.61$, $M = 0.32$, $\sigma = 0.78$) routes.

were $\approx 0m/s^2$ for every route. Deriving the above limits from the probability distribution of several users also helps eliminate privacy leaks because these limits cannot then be used to identify the driving behavior of a specific user, thus, masking all the users.

The above constraints can be formulated as a list of equalities and inequalities, this time defining a nonlinear constraint optimization problem. Such problems can be solved by using Sequential Quadratic Programming (SQP) methods. Let $a = (a_1, \ldots, a_N)$ denote a vector of acceleration values for each step, where N denotes the travel time of the step, i.e., $N = int(t_{step})$. Let v_0 denote the initial speed coming into this step and $v = (v_1, \ldots, v_N)$ denote a vector of speeds calculated from v_0 and the vector a. The objective of this optimization is to find an optimal vector a that minimizes $|\bar{v} - (d_{step}/t_{step})| < \Delta$ to adhere to the traffic constraints, where $\bar{v}$ is the mean of vector v, and d_{step}, t_{step} represent the step's distance and time. The Δ is a threshold that determines whether the minimized objective function value is acceptable. All rejected optimizations are retried with a higher number of iterations till a valid solution satisfying the threshold is found. We observed that this optimization typically yields an optimal vector a that approaches the lower mean bound of the absolute accelerations $|\bar{a}|_{min}$, for most optimizations. To address this, we derive a new lower mean bound for every route from a uniform distribution and use the following range for optimization: $[|\bar{a}|_{rand}, |\bar{a}|_{max}]$, where $|\bar{a}|_{rand} = \mathcal{U}(|\bar{a}|_{min}, |\bar{a}|_{max} - \delta)$, and δ is a small constant to ensure that $|\bar{a}|_{rand} < |\bar{a}|_{max}$. The optimal vectors a_i for every step i are merged to represent the route's accelerations. Note that a bounded constraint of the form $x_1 \leq x \leq x_2$ can be rewritten as $(x_2 - x)(x - x_1) \geq 0$ for simplifying the constraint for the solver. Using above attributes, the route optimization for each step is formally defined as

$$
\begin{aligned}
\text{Minimize} \quad & |\bar{v} - (d_{step}/t_{step})| \\
\text{Subject to:} \quad & \bar{a} = 0 \\
& (|\bar{a}|_{max} - |\bar{a}|)(|\bar{a}| - |\bar{a}|_{rand}) \geq 0 \\
& (\sigma(|a|)_{max} - \sigma(|a|))(\sigma(|a|) - \sigma(|a|)_{min}) \geq 0 \\
& \sigma(|a|) - |\bar{a}| \geq 0 \\
\text{Bounds:} \quad & a_{min} \leq a_j \leq a_{max} \qquad \text{for } j = 1, 2, \ldots, N
\end{aligned}
$$

Some additional constraints applied to the optimization are that $v_0 = 0$ for the first step and $v_N = 0$ for the last step of the route. The optimization is improved by providing an initial guess of bounded accelerations from a gaussian distribution $\mathcal{N}(\bar{v'}, 2)$, where $\mu = \bar{v'}$ is the mean step speed, i.e., $\bar{v'} = d_{step}/t_{step}$, and $\sigma = 2m/s$ is the standard deviation of the speed. Figure 7b shows the distribution of the absolute accelerations generated for synthetic trajectories. We can observe that the parameters and shape of the distribution closely follows the parameters and shape of the real distribution. As GPS accuracy varies, a small random gaussian noise is added to each coordinate of the final trajectory.

Synthesizing the sensor route between two states: The sensor data (Accelerometer, Gyroscope and Magnetometer) can be synthesized between two states using the GPS trajectory generated in the previous step. By modeling the synthetic sensor data on GPS trajectories, we ensure that the generated sensor traces follow the accelerations, left/right turns, and curvature of the GPS trajectory. We further add noise with probability distributions from real sensor measurements on modern smartphones. This is done to simulate noise in the sensor data (e.g., Gyroscope reports 97 degree turn when the actual turn was 90).

To generate synthetic sensor trajectories, we first assume that the phone is perfectly aligned to the vehicle's reference frame while driving. This simplifies calculating the synthetic sensor values. The assumption is later relaxed to support different phone orientations. For the Accelerometer, in the above orientation, the y axis will experience all the vehicular accelerations, z axis will experience the gravitational force (i.e., 9.8) and x axis will be zero. For the Gyroscope, the z axis will experience all the vehicular rotations and the x and y axes will be zero. For the Magnetometer, the z axis will be zero while the x and y axes will report values based on the vehicle bearing with respect to magnetic north. Note that these values can be pre-calculated and loaded based on the desired bearing. Let $a = (a_0, \ldots, a_N)$ denote a vector of acceleration values and $b = (b_0, \ldots, b_N)$ denote a vector of bearings derived from the geographic

coordinates of the synthetic GPS trajectory. At any time instant $t \in N$, the Accelerometer sample will be $(n_{ax}, a_t + n_{ay}, n_{az} + 9.8)_t$, the Gyroscope sample will be $(n_{gx}, n_{gy}, b_t - b_{(t-1)} + n_{gz})_t$, and the Magnetometer sample will be $(x + n_{mx}, y + n_{my}, n_{mz})_t$. Here, n represents a random noise added to each sensor axis whose limits are derived by performing a calibration step. During calibration, the user places the phone on a flat surface for a short duration and the Accelerometer, Magnetometer bias and Gyroscope drift is recorded on all axes. The sensor data can now be easily rotated into different orientations (e.g., phone on a mount, in pocket) by calculating a rotation matrix for that orientation and using it to rotate each individual sample.

5.3 Limitations and Future Work

In this section, we discuss some limitations and privacy utility trade-offs of the Synthetic Location Provider.

Synthesizing GPS Trajectories: This work uses a linear model for synthesizing walks from a state's coordinates to a graph vertex, and vice versa. The vertex containing a point nearest to the state's coordinates is chosen, and the driving route is started/stopped at this point. This simple model assumes a constant walking speed as our main focus was on driving. We plan to study models for generating realistic walk patterns in the future in our continued effort to improve MATRIX.

Synthesizing Sensor Trajectories: The sensor obfuscation technique solves the problem of sensor obfuscation when the user is driving, but what about when the user is walking or using the phone for games or browsing apps? We believe that obfuscating such scenarios requires a thorough analysis of the sensor patterns when such behaviors occur. There is also the need to analyze what type of approaches (e.g., discriminative, generative [65], generative adversarial [36]) are suitable for modeling this behavior. This is beyond the scope of the current work and we intend to address this problem in the future.

Privacy Utility Trade-offs: MATRIX provides smartphone users a tool to feed obfuscated or synthetic locations to apps that access a user's precise location, but don't need this information to function. Such over-provisioned apps are exemplified by "Brightest Flashlight" that accesses users' location when started, yet does not provide location-based services [33]. The goal is to give users the flexibility of synthesizing their location information for such apps, while still use other location-based apps by setting them to obfuscated / high accuracy options. Location-based apps that use approximate location (e.g., TripAdvisor, Yelp, Weather Channel) can still function as intended by setting their accuracy to block level obfuscation. Other apps that require more precise information (e.g., Google Navigation, Waze) can function properly only by setting them to high accuracy. Currently, MATRIX does not provide the capability of preserving a users' driving profile such as their speeds and distances while synthesizing their location. Such capability could be added by mapping the user's real route to a synthetic route in *real-time*, and moving the synthetic identity with the same speed on the synthetic route. In case the streets of the routes are of different lengths and the real user makes a turn, the synthetic identity can keep going straight at the user's speed and take the next turn or not. The sensor data can also be obfuscated in real-time using the GPS coordinates of the route. While such an approach will not preserve the turns profile (which can reveal the real location of users [62]), it can preserve the traveled distance and accelerations pattern (which might be useful in some contexts such as monitoring a car's health as a function of traveled distance). We intend to carefully design and implement such an extension in the future. We re-emphasize that MATRIX was designed to be extensible and already existing trajectory generators that provide similar capabilities (e.g., [15, 32]) can be easily integrated and used as alternative Location Providers.

Wireless Tracking: MATRIX does not defend against wireless tracking attacks that perform fingerprinting on the device RF radio interface [79], or de-anonymize the authentication protocol [19]. However, such attacks are harder to achieve as they require the adversary to be in the vicinity of the target or have control over the authentication infrastructure.

6 Evaluation

In this section, we evaluate MATRIX using the following metrics: the portability and performance of the PrivoScope and Synthetic Location services, the utility of the PrivoScope service based on a user study, and the detection of synthetic trajectories by popular location-driven apps and Machine Learning algorithms. We also include an evaluation of the detection of synthetic trajectories by regular users (cf. Appendix A) and the system stability (cf. Appendix B) in the Appendix due to paper length requirements.

6.1 System Portability and Performance

MATRIX is compatible with Android `KitKat` and onwards. It has been tested to work on Xposed Framework API versions 82 to 89 (current) which are compatible with the above Android versions. This implies that MATRIX can be ported to ≈ 94% of all Android devices globally (based on information from the Android Dashboard [8] as of August 25, 2018).

MATRIX was extensively evaluated for performance overheads occurring from the most expensive operations of the system. We identified 3 potential performance bottlenecks in our system: (1) the API call interception function using the Xposed framework; (2) the add audit event function of the PrivoScope service; and (3) the location provider function of the Synthetic Location service. We implemented a test app that invoked these functions 1 million times to test performance. The execution time was calculated as the difference between two `System.nanoTime` method calls placed immediately before and after the function execution. The API interception bottleneck is caused by the Xposed framework loading and hooking method calls. To evaluate its performance, we created an empty method inside our system and hooked it using the Xposed framework.

Table 1 shows the mean μ, standard deviation σ and maximum time of execution for the three functions on a LG Nexus 5 and a LG Nexus 5X. The API interception function using the Xposed framework averaged to $\mu = 0.2ms$ on both the phones, which is negligible from a usage perspective. The add audit event function of PrivoScope had a low μ for both the phones ($4.3ms$ and $3.2ms$, resp), and its performance is also acceptable. The location provider function of the Synthetic Location service had a relatively higher μ and σ for the Nexus 5 ($\mu = 11.1ms$, $\sigma = 7.7ms$). We believe this overhead is due to database lookups performed by the service to check the location preferences for the app. Overall, the entire system can run with an average overhead of $15.6ms$ on the Nexus 5 and $9.1ms$ on the Nexus 5X which should have a negligible impact on the user experience. The sum of worst case performances overhead at $171.8ms$ on the Nexus 5 should also not affect user experience since such overhead occurs rarely.

Table 1. Results of the Performance evaluation of MATRIX for 2 smartphones.

Phone	Service	Mean (μ)	Std (σ)	Max
Nexus 5	Xposed Hook	0.2 ms	0.3 ms	17.1 ms
	Add Audit Event	4.3 ms	3.8 ms	67.1 ms
	Update Location	11.1 ms	7.7 ms	87.6 ms
Nexus 5X	Xposed Hook	0.2 ms	0.15 ms	5.7 ms
	Add Audit Event	3.2 ms	1.6 ms	26.8 ms
	Update Location	5.7 ms	1.5 ms	16.0 ms

6.2 Utility of the PrivoScope Service

To evaluate the utility of the PrivoScope service, we conducted a user study for a group of 15 participants from different educational backgrounds. Of these participants, 5 are in a non-CS profession (3 doctors, 1 chemist and 1 lawyer), 5 are software engineers with some security knowledge, and the remaining are security and privacy researchers. The user study steps were as follows: (1) we asked the users how knowledgeable they were about location tracking from the GPS, WiFi, cellular, and the sensors, (2) we discussed some examples of location and sensor tracking with them to spread awareness of these threats to their privacy, (3) we gave them an introduction of MATRIX and then presented a demo[1] of the PrivoScope service, (4) we asked the participants to use three flashlight apps on a Nexus 5 to assess their comfort with using the device and the PrivoScope service, and (5) we asked them to answer some questions regarding PrivoScope using a Google form (see Appendix C for the questions and responses).

The device under test was preloaded with common apps (e.g., Bank of America, Google Maps, Uber, Facebook, Twitter, etc). It was used as a regular device for ≈ 2 weeks to collect location and sensor data for the demo. We found some interesting data such as the Bank of America app querying location data once every 3-4 hours. We believe they implement some form of user protection using this data. We also observed that Google Maps has a fixed pattern for accessing location data. Such patterns can possibly be used to create app signatures and detect situations when a clone app from a third party app store is installed instead. Discussing all observations is outside the scope of this paper. Regarding the flashlight apps, App1 (flashlight) was used as it did not access location or sensors, App2 (Flashlight Plus) accessed the Accelerometer when in foreground and ran as a background service all the time, and App3 (Brightest Flashlight) accessed the location of the users when in foreground.

The results of this user study were encouraging. We refer readers to Appendix C for all responses. To summarize the results, 12 users found PrivoScope to be very

1 A video of the demo is available at https://youtu.be/GkboLYZfLcA

useful, and 3 users found it to be useful. They mentioned that adding a summary for all apps will make the tool very useful, and we plan to incorporate this in PrivoScope before releasing the source code. All of them responded positively about recommending PrivoScope to their friends and family. Regarding the flashlight apps, many users initially preferred App2 as it looked nicer, but all chose App1 after looking at their usages in PrivoScope. This also validates the analysis of [2] that suggested that users are reluctant to change their privacy settings unless they are nudged in some manner. All except one, mentioned that they're not comfortable with App2 running in the background and accessing any form of their personal information. One person chose App1 as it would cause less battery drain. Since the study, 3 participants have started using MATRIX on their devices and we hope to soon make this tool available globally.

6.3 Detection of Synthetic Trajectories

Detection by Popular Mobile Apps: We evaluated this metric using 10 popular location-driven apps (listed in Table 2) on Google Play Store. We extend the evaluation of [21] who showed that the top 300 apps on the Play Store do not crash when supplied with a fake location. Their approach focused on a single location fix which may not provide these apps enough information to detect the presence of synthetic feeds. The apps we choose rely heavily on location data to provide their services to users. The evaluation was performed by feeding these apps three types of synthetic location data and monitoring their behavior. In test 1 **(Synthetic)**, the synthetic trajectories were generated using the techniques described in Section 5.2. In test 2 **(HS)**, the trajectories from test 1 were time compressed by a factor of 5 such that the user appeared to move 5 times faster (e.g., at $300km/h$ in a $60km/h$ speed zone). In test 3 **(HS+T)**, the trajectories from test 2 were perturbed by large noises ($\approx 1000m$) such that the user appeared to teleport to different locations very quickly. The expected results was that apps that detect fake location should be able to easily detect the **HS** and **HS+T** trajectories, but not **Synthetic** trajectories.

Table 2 shows the results of the three tests for our test apps. None of the apps were able to detect synthetic locations in the **Synthetic** trajectories test. Even for **HS** and **HS+T** trajectories test, with the exception of Ingress, none of the other apps detected the presence of high speed and noisy synthetic locations. Ingress did not ban us from playing the game, however, it denied points when it detected that the user was moving too fast or teleporting. Pokémon Go is also known to ban users, however, we did not get banned during our tests even after capturing many Pokémons using the noisy data. This is likely because the ban threshold is set to high to prevent users from going to a higher level by cheating. All the remaining apps kept performing their functions without detecting the presence of the synthetic data. Note that Waze and Google Maps navigation operated properly for **HS** but became unstable for **HS+T**, which was expected as they constantly updated the routing information based on the teleported locations.

Table 2. Results of the Synthetic Trajectories detection test on 10 popular Android apps that rely on location data.

App Name	Rating	Synthetic	High Speed (HS)	HS+Teleport (HS+T)
Ingress	4.3	✓	Detected	Detected
Pokémon Go	4.1	✓	✓	✓
Geocaching	4.0	✓	✓	✓
Glympse	4.5	✓	✓	✓
Family Locator	4.4	✓	✓	✓
happn	4.5	✓	✓	✓
Yelp	4.3	✓	✓	✓
Foursquare	4.1	✓	✓	✓
Waze	4.6	✓	✓	Unstable
Google Maps	4.3	✓	✓	Unstable

These observations indicate that popular location-driven apps fail to check validity of the received data. Some of these apps (Ingress, Pokémon Go, Foursquare and Google Maps) check whether the `MockLocationProvider` [5] is enabled on the device. Some apps rely on other schemes to limit user abuse (e.g., Foursquare detects and limits rapid check-ins). This means that they rely on simple checks but do not implement algorithms for detecting synthetic data. The only app that checked location validity in our set was Ingress, and it was unable to detect any discrepancies in the synthetic trajectories generated by our system.

Detection by Machine Learning Algorithms: We evaluated this metric using the 400 routes collected for analyzing user driving behavior (cf. Section 5.2). These set of routes were labeled as 'Real'. For each real route, a corresponding synthetic route was generated using the real route's departure time, and start and end locations. These set of routes were labeled as 'Synthetic'. As different routes may contain completely different trajectories, a machine learning model comprising of spatial features of the routes will be very inaccurate. Therefore, we focus on the temporal features of these routes, namely the accelerations, speeds and distances traveled. Recall that the routes were driven on different road types (highways

Table 3. Results of the Machine Learning algorithms evaluation showing accuracy of 'Real' trajectories detected as 'Real', 'Synthetic' trajectories detected as 'Synthetic', and the average Precision, Recall and F-score values.

Algorithm	Real	Synthetic	Precision	Recall	F-score
Decision Trees	53%	62%	0.576	0.575	0.574
Random Forest	61%	63%	0.620	0.620	0.619
Nearest Neighbor	50%	57%	0.535	0.535	0.534
10 Nearest Neighbor	49%	57%	0.530	0.530	0.529
Naive Bayes	86%	14%	0.500	0.500	0.426
Neural Networks	95%	4%	0.471	0.495	0.363
SVM	5%	97%	0.565	0.510	0.378

and internal) and traffic conditions (cf. Section 5.2) indicating high variation in the distances traveled and driving speeds. As a result, the strongest features are those derived from the accelerations. To validate this analysis, we evaluated different features using Information Gain (IG) values that tree-based classification algorithms like Decision Trees use for attribute splitting. Higher IG values indicate that a feature is more important for machine learning classification. After many experiments, the following 9 features were selected as they yielded the highest information gain values and prediction accuracies among other combinations: *max (IG*=0.07*) and min (IG*=0.07*) accelerations*, *mean (IG*=0.03*) and standard deviation (IG*=0.21*) of accelerations*, *mean (IG*=0.08*) and standard deviation (IG*=0.26*) of absolute accelerations*, *maximum speed (IG*=0.11*)*, *idle time (IG*=0.09*)* and *distance traveled (IG*=0.08*)*. The above weighted information gain values show that the *standard deviation of accelerations and absolute accelerations* are the strongest in determining the distinction between synthetic and real traces. These values were derived using Scikit-learn [70] toolkit in Python. The models were built and the predictions were averaged over 1000 iterations. In each iteration, 90% of the dataset from each set were randomly chosen for training data, and the remaining 10% from each set were test data.

Table 3 shows the list of algorithms that were evaluated, the prediction accuracies for 'Real' test trajectories that were detected as 'Real', 'Synthetic' test trajectories that were detected as 'Synthetic', and the Precision, Recall and F-score values. Note that in our context, the results for an ideal MATRIX route generator should lead to a 50-50 split, i.e., 50% of 'Real' routes are predicted as 'Synthetic' and 50% of 'Synthetic' routes are predicted as 'Real'. We can observe that most algorithms (except Decision Trees) have an average prediction accuracy of close to 50%. Three of those algorithms (Naive Bayes, Neural Network and SVM) display results biased towards one of the two classifiers implying that the models had difficulty predicting the correct classifier and defaulted to one classifier. The Decision Trees algorithm could detect $\approx 62\%$ of the 'Synthetic' trajectories as synthetic. The ensemble classifier of Decision Trees, Random Forest, could detect $\approx 63\%$ of the 'Synthetic' trajectories as synthetic. These numbers also do not signify large detection rate for our synthetic trajectories. We must note that this evaluation is preliminary as 800 routes do not suffice for these algorithms to build generalized models from training data, and the models may be subject to overfitting. We intend to extend this dataset in the future to incorporate more routes and run this evaluation again for more generalized models.

7 Related Work and Discussion

A large body of research has focused on mitigating location and other private information leakage attacks on Android devices. Most of these works are orthogonal to our system as their motivation and techniques differ, including recommending new frameworks/privacy metrics [12, 14, 27, 31, 32, 52, 69], location obfuscation [4, 9, 18, 73, 80], location cloaking [38, 41], generating dummy locations [48, 49, 56, 75, 85, 87], sensor data obfuscation [24, 25], tainting sensitive data [30, 74], dynamic analysis [83, 90], static code analysis [10, 11, 57, 81], permissions analysis [46], and application retrofitting [26, 43, 91].

Synthesizing human mobility was also studied in the context of opportunistic networks [29, 35, 47, 53], ad-hoc and vehicular wireless networks [22, 50, 78, 88], community based mobility models [39, 42, 54, 61], predicting location of moving objects [76, 89], and implementing efficient location update mechanisms [28, 44, 45, 82]. Some research also focused on generating synthetic traces for user privacy [15, 23, 51], however, these works have limitations that enable an adversary to detect fake traces. None of these satisfy traffic constraints for different roads at different times of the day, nor take into account the statistical properties of user driving behavior. For example, [51, 82] simply superimpose speed patterns from real routes on synthetic traces based on the street type without accounting for traffic conditions of the road. These speed patterns can be repeated and can be detected.

Beresford et al. [13] implemented MockDroid, a modified version of Android 2.2.1 with a user controlled permissions manager. The system allowed users to define mock permissions for installed apps. The location

mock permission was implemented to block all location fixes from reaching the app simulating a lack of available location information. The authors ran the system on 27 apps and showed that most apps continued to function with reduced functionality. This work is similar to the current Android permissions model and, therefore, subject to the weaknesses in Android's permission model that we have addressed with MATRIX.

Agarwal and Hall [1] implemented ProtectMyPrivacy (PMP) for iOS jailbroken devices that intercepted method calls accessing user's private data, and allowed the user to substitute anonymized data in place of the real information. The system was later implemented for Android [21]. The limitations of these work are: (1) the anonymized data is provided by the user at run-time which may be completely random and unrealistic, and (2) the app's functionality is paused for user input which is detrimental to user experience and possibly to the app's functionality. The above limitations are addressed by MATRIX through a seamless delivery of realistic synthetic locations to apps without user interaction.

Liu et al. [55] implemented Personalized Privacy Assistant (PPA) for rooted Android devices. This system is a modified `App Ops` permission manager that displays an app's recent requests and the frequencies of requests in the past 7 days. The system uses this information to generate daily privacy nudges to motivate users to interact and change their privacy settings. Similarly, Fu et al. [34] implemented a system for Android that displays a user notification when an installed app accessed the user location, and showed this location on Google Maps. However, additional context is required to determine if an app is misusing the information (e.g., time and duration of requests, was the app in the background?). PrivoScope addresses the limitations by providing much more context to the users and displaying them in a way that it is easier for the users to grasp and visualize accesses.

Zheng et al. [88] propose an agenda driven mobility model that considers a person's daily social activities for motion generation. They derive this agenda from the National Household Travel Survey (NHTS) database by the U.S. Department of Transportation. The first agenda and all subsequent activities are based on the NHTS activity distribution, and addresses are picked at random from many addresses for the corresponding activity. The start time of the first agenda determines the schedule for the entire day and each activity starts immediately after the mean dwell time+longest transition time from previous activity. The route between two activities assumes a longest possible time given by the Dijkstra's algorithm. This work has several limitations (all addressed by MATRIX) that are trivial to detect: (1) the addresses are picked at random without accounting for distances (e.g., gas station may be miles away from regular route), (2) the routes do not incorporate any traffic information and are always static, and (3) the routes do not incorporate any driving behavior and likely assume a constant speed of motion.

Bindschaedler and Shokri [15] generate synthetic traces that are derived from seed datasets of real traces. We believe that this work is orthogonal to our work and potentially complementary. MATRIX focuses on solving the problem of creating identities, user schedules, and movement patterns; and seamlessly integrating them in Android. Our trajectory generation scheme focuses on adhering to daily schedule tasks/time constraints, traffic, and generating acceleration patterns similar to real user driving behavior. We chose an alternative approach to them as they rely on data-sets of real traces which limits scalability to cities where traces are unavailable.

Fawaz and Shin [32] implemented LP-Guardian, a privacy protection framework modifying the Android source code. The framework changes location granularity of installed apps based on the threat posed by the app and its location granularity requirements. It automatically coarsens the location to a city level if it identifies a request from an A&A library, the app is in the background, or the app is a weather app. It synthesizes the location for fitness apps but preserves features of the actual route such as the distance traveled. The framework supplies a synthetic location if it determines that it is not safe to release the location. This work has the following limitations that are addressed in MATRIX: (1) the preservation of route features can lead to inference of the user's real locations, and (2) unless chosen very carefully, the synthetic traces generated from real features will not snap to streets (e.g., different street lengths and curvatures) and can be detected as synthetic.

In the future, we plan to extend MATRIX with models that emulate human walks for location and sensor obfuscation, as well as models for sensor obfuscation that precisely emulate user behavior when they interact with their smartphones.

8 Acknowledgments

We would like to thank Dr. Thomas Roessler for his helpful comments. This material is based upon work partially supported by the National Science Foundation under Grant No. 1740907.

References

[1] Yuvraj Agarwal and Malcolm 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*, MobiSys '13, New York, NY, USA, 2013. ACM.

[2] Hazim Almuhimedi, Florian Schaub, Norman Sadeh, Idris Adjerid, Alessandro Acquisti, Joshua Gluck, Lorrie Faith Cranor, and Yuvraj Agarwal. Your location has been shared 5,398 times!: A field study on mobile app privacy nudging. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, CHI '15, New York, NY, USA, 2015. ACM.

[3] Amazon. Amazon Mechanical Turk. https://www.mturk.com/mturk/welcome, 2017.

[4] Miguel E. Andrés, Nicolás E. Bordenabe, Konstantinos Chatzikokolakis, and Catuscia Palamidessi. Geo-indistinguishability: Differential privacy for location-based systems. In *Proceedings of the 2013 ACM SIGSAC Conference on Computer & Communications Security*, CCS '13, 2013.

[5] Android. Android Mock Location Provider. https://developer.android.com/guide/topics/location/strategies.html#MockData, 2017.

[6] Android. The Android Source Code. https://source.android.com/source/, 2017.

[7] Android. UI/Application Exerciser Monkey. https://developer.android.com/studio/test/monkey.html, 2017.

[8] Android. Android Dashboards. https://developer.android.com/about/dashboards/index.html, 2018.

[9] C. A. Ardagna, M. Cremonini, S. De Capitani di Vimercati, and P. Samarati. An obfuscation-based approach for protecting location privacy. *IEEE Transactions on Dependable and Secure Computing*, Jan 2011.

[10] Steven Arzt, Siegfried Rasthofer, Christian Fritz, Eric Bodden, Alexandre Bartel, Jacques Klein, Yves Le Traon, Damien Octeau, and Patrick McDaniel. Flowdroid: Precise context, flow, field, object-sensitive and lifecycle-aware taint analysis for android apps. In *Proceedings of the 35th ACM SIGPLAN Conference on Programming Language Design and Implementation*, PLDI '14, New York, NY, USA, 2014. ACM.

[11] Michael Backes, Sven Bugiel, Erik Derr, Sebastian Gerling, and Christian Hammer. R-droid: Leveraging android app analysis with static slice optimization. In *Proceedings of the 11th ACM on Asia Conference on Computer and Communications Security*, ASIA CCS '16, New York, NY, USA, 2016. ACM.

[12] Michael Backes, Sven Bugiel, Christian Hammer, Oliver Schranz, and Philipp von Styp-Rekowsky. Boxify: Full-fledged app sandboxing for stock android. In *24th USENIX Security Symposium (USENIX Security 15)*, Washington, D.C., 2015. USENIX Association.

[13] Alastair R. Beresford, Andrew Rice, Nicholas Skehin, and Ripduman Sohan. Mockdroid: Trading privacy for application functionality on smartphones. In *Proceedings of the 12th Workshop on Mobile Computing Systems and Applications*, HotMobile '11, New York, NY, USA, 2011. ACM.

[14] Igor Bilogrevic, Kévin Huguenin, Berker Agir, Murtuza Jadliwala, Maria Gazaki, and Jean-Pierre Hubaux. A machine-learning based approach to privacy-aware information-sharing in mobile social networks. *Pervasive and Mobile Computing*, 25, 2016.

[15] V. Bindschaedler and R. Shokri. Synthesizing plausible privacy-preserving location traces. In *2016 IEEE Symposium on Security and Privacy (SP)*, May 2016.

[16] Kenneth Block, Sashank Narain, and Guevara Noubir. An autonomic and permissionless android covert channel. In *Proceedings of the 10th ACM Conference on Security and Privacy in Wireless and Mobile Networks*, WiSec '17, 2017.

[17] Kenneth Block and Guevara Noubir. My magnetometer is telling you where i've been?: A mobile device permissionless location attack. In *Proceedings of the 11th ACM Conference on Security & Privacy in Wireless and Mobile Networks*, WiSec '18, 2018.

[18] Nicolás E. Bordenabe, Konstantinos Chatzikokolakis, and Catuscia Palamidessi. Optimal geo-indistinguishable mechanisms for location privacy. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, CCS '14, 2014.

[19] Aldo Cassola, Erik-Oliver Blass, and Guevara Noubir. Authenticating privately over public wi-fi hotspots. In *Proceedings of the 22Nd ACM SIGSAC Conference on Computer and Communications Security*, CCS '15, 2015.

[20] Aldo Cassola, William Robertson, Engin Kirda, and Guevara Noubir. A Practical, Targeted, and Stealthy Attack Against WPA Enterprise Authentication. In *NDSS Symposium 2013*, 2013.

[21] Saksham Chitkara, Nishad Gothoskar, Suhas Harish, Jason I. Hong, and Yuvraj Agarwal. Does this app really need my location?: Context-aware privacy management for smartphones. *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.*, September 2017.

[22] David R. Choffnes and Fabián E. Bustamante. An integrated mobility and traffic model for vehicular wireless networks. In *Proceedings of the 2Nd ACM International Workshop on Vehicular Ad Hoc Networks*, VANET '05, 2005.

[23] Richard Chow and Philippe Golle. Faking contextual data for fun, profit, and privacy. In *Proceedings of the 8th ACM Workshop on Privacy in the Electronic Society*, WPES '09, 2009.

[24] Anupam Das, Nikita Borisov, and Matthew Caesar. Tracking mobile web users through motion sensors: Attacks and defenses. In *Network and Distributed System Security Symposium (NDSS)*, 2016.

[25] Anupam Das, Nikita Borisov, and Edward Chou. Every move you make: Exploring practical issues in smartphone motion sensor fingerprinting and countermeasures. *Proceedings on Privacy Enhancing Technologies (PoPETs)*, pages 88–108, 2018.

[26] Benjamin Davis and Hao Chen. Retroskeleton: Retrofitting android apps. In *Proceeding of the 11th Annual International Conference on Mobile Systems, Applications, and Services*, MobiSys '13, New York, NY, USA, 2013. ACM.

[27] B. Deva, S. R. Garzon, and S. Schünemann. A context-sensitive privacy-aware framework for proactive location-based services. In *2015 9th International Conference on Next Generation Mobile Applications, Services and Technologies*,

Sept 2015.

[28] Z. Ding, L. Guo, and X. Meng. Adaptive location update mechanism for network-constrained moving objects in changeful traffic conditions. In *2009 Tenth International Conference on Mobile Data Management: Systems, Services and Middleware*, May 2009.

[29] Frans Ekman, Ari Keränen, Jouni Karvo, and Jörg Ott. Working day movement model. In *Proceedings of the 1st ACM SIGMOBILE Workshop on Mobility Models*, MobilityModels '08, 2008.

[30] William Enck, Peter Gilbert, Byung-Gon Chun, Landon P. Cox, Jaeyeon Jung, Patrick McDaniel, and Anmol N. Sheth. Taintdroid: An information-flow tracking system for real-time privacy monitoring on smartphones. In *Proceedings of the 9th USENIX Conference on Operating Systems Design and Implementation*, OSDI'10, Berkeley, CA, USA, 2010. USENIX Association.

[31] Kassem Fawaz, Huan Feng, and Kang G. Shin. Anatomization and protection of mobile apps' location privacy threats. In *24th USENIX Security Symposium (USENIX Security 15)*. USENIX Association, 2015.

[32] Kassem Fawaz and Kang G. Shin. Location privacy protection for smartphone users. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, CCS '14, 2014.

[33] FTC. Android flashlight app developer settles FTC charges it deceived consumers. https://www.ftc.gov/news-events/press-releases/2013/12/android-flashlight-app-developer-settles-ftc-charges-it-deceived, December 2013. Accessed: November, 2015.

[34] Huiqing Fu, Yulong Yang, Nileema Shingte, Janne Lindqvist, and Marco Gruteser. A field study of run-time location access disclosures on android smartphones. In *Network and Distributed System Security Symposium (NDSS)*, 2014.

[35] J. Ghosh, S. J. Philip, and C. Qiao. Sociological orbit aware location approximation and routing in manet. In *2nd International Conference on Broadband Networks, 2005.*, Oct 2005.

[36] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In Z. Ghahramani, M. Welling, C. Cortes, N. D. Lawrence, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 27*, pages 2672–2680. 2014.

[37] Jun Han, E. Owusu, L.T. Nguyen, A. Perrig, and J. Zhang. Accomplice: Location inference using accelerometers on smartphones. In *Communication Systems and Networks (COMSNETS), 2012 Fourth International Conference on*, Jan 2012.

[38] B. Henne, C. Kater, M. Smith, and M. Brenner. Selective cloaking: Need-to-know for location-based apps. In *2013 Eleventh Annual Conference on Privacy, Security and Trust*, July 2013.

[39] Klaus Herrmann. Modeling the sociological aspects of mobility in ad hoc networks. In *Proceedings of the 6th ACM International Workshop on Modeling Analysis and Simulation of Wireless and Mobile Systems*, MSWIM '03, 2003.

[40] Baik Hoh, M. Gruteser, Hui Xiong, and A. Alrabady. Enhancing security and privacy in traffic-monitoring systems. *IEEE Pervasive Computing*, Oct 2006.

[41] Baik Hoh and Marco Gruteser. Preserving privacy in gps traces via uncertainty-aware path cloaking. In *In Proceedings of ACM CCS 2007*, 2007.

[42] Xiaoyan Hong, Mario Gerla, Guangyu Pei, and Ching-Chuan Chiang. A group mobility model for ad hoc wireless networks. In *Proceedings of the 2Nd ACM International Workshop on Modeling, Analysis and Simulation of Wireless and Mobile Systems*, MSWiM '99, 1999.

[43] Peter Hornyack, Seungyeop Han, Jaeyeon Jung, Stuart Schechter, and David Wetherall. These Aren'T the Droids You'Re Looking for: Retrofitting Android to Protect Data from Imperious Applications. In *Proceedings of the 18th ACM Conference on Computer and Communications Security*, CCS '11, New York, NY, USA, 2011. ACM.

[44] Y. K. Huang, I. F. Su, L. F. Lin, and Y. C. Chung. Efficient processing of updates for moving objects with varying speed and direction. In *2013 IEEE 27th International Conference on Advanced Information Networking and Applications (AINA)*, March 2013.

[45] Yuan-Ko Huang. Indexing and querying moving objects with uncertain speed and direction in spatiotemporal databases. *Journal of Geographical Systems*, Apr 2014.

[46] Jinseong Jeon, Kristopher K. Micinski, Jeffrey A. Vaughan, Ari Fogel, Nikhilesh Reddy, Jeffrey S. Foster, and Todd Millstein. Dr. android and mr. hide: Fine-grained permissions in android applications. In *Proceedings of the Second ACM Workshop on Security and Privacy in Smartphones and Mobile Devices*, SPSM '12, New York, NY, USA, 2012. ACM.

[47] D. Karamshuk, C. Boldrini, M. Conti, and A. Passarella. Human mobility models for opportunistic networks. *IEEE Communications Magazine*, December 2011.

[48] Ryo Kato, Mayu Iwata, Takahiro Hara, Akiyoshi Suzuki, Xing Xie, Yuki Arase, and Shojiro Nishio. A dummy-based anonymization method based on user trajectory with pauses. In *Proceedings of the 20th International Conference on Advances in Geographic Information Systems*, SIGSPATIAL '12, 2012.

[49] H. Kido, Y. Yanagisawa, and T. Satoh. An anonymous communication technique using dummies for location-based services. In *ICPS '05. Proceedings. International Conference on Pervasive Services, 2005.*, July 2005.

[50] M. Kim, D. Kotz, and S. Kim. Extracting a mobility model from real user traces. In *Proceedings IEEE INFOCOM 2006. 25TH IEEE International Conference on Computer Communications*, April 2006.

[51] John Krumm. Realistic driving trips for location privacy. In *International Conference on Pervasive Computing*. Springer, 2009.

[52] B. Krupp, N. Sridhar, and W. Zhao. Spe: Security and privacy enhancement framework for mobile devices. *IEEE Transactions on Dependable and Secure Computing*, 2015.

[53] K. Lee, S. Hong, S. J. Kim, I. Rhee, and S. Chong. Slaw: A new mobility model for human walks. In *IEEE INFOCOM 2009*, April 2009.

[54] Baochun Li. On increasing service accessibility and efficiency in wireless ad-hoc networks with group mobility. *Wirel. Pers. Commun.*, April 2002.

[55] Bin Liu, Mads Schaarup Andersen, Florian Schaub, Hazim Almuhimedi, Shikun (Aerin) Zhang, Norman Sadeh, Yuvraj Agarwal, and Alessandro Acquisti. Follow my recommen-

dations: A personalized privacy assistant for mobile app permissions. In *Twelfth Symposium on Usable Privacy and Security (SOUPS 2016)*, Denver, CO, 2016. USENIX Association.

[56] Hua Lu, Christian S. Jensen, and Man Lung Yiu. Pad: Privacy-area aware, dummy-based location privacy in mobile services. In *Proceedings of the Seventh ACM International Workshop on Data Engineering for Wireless and Mobile Access*, MobiDE '08, 2008.

[57] Kangjie Lu, Zhichun Li, Vasileios P. Kemerlis, Zhenyu Wu, Long Lu, Cong Zheng, Zhiyun Qian, Wenke Lee, and Guofei Jiang. Checking more and alerting less: Detecting privacy leakages via enhanced data-flow analysis and peer voting. In *The Network and Distributed System Security Symposium*, NDSS '15, 2015.

[58] Ashwin Machanavajjhala, Daniel Kifer, John Abowd, Johannes Gehrke, and Lars Vilhuber. Privacy: Theory meets practice on the map. In *Proceedings of the 2008 IEEE 24th International Conference on Data Engineering*, ICDE '08, 2008.

[59] Yan Michalevsky, Aaron Schulman, Gunaa Arumugam Veerapandian, Dan Boneh, and Gabi Nakibly. Powerspy: Location tracking using mobile device power analysis. In *24th USENIX Security Symposium (USENIX Security 15)*. USENIX Association, 2015.

[60] A. Mosenia, X. Dai, P. Mittal, and N. Jha. Pinme: Tracking a smartphone user around the world. *IEEE Transactions on Multi-Scale Computing Systems*, 2017.

[61] Mirco Musolesi and Cecilia Mascolo. A community based mobility model for ad hoc network research. In *Proceedings of the 2Nd International Workshop on Multi-hop Ad Hoc Networks: From Theory to Reality*, REALMAN '06, 2006.

[62] S. Narain, T. D. Vo-Huu, K. Block, and G. Noubir. Inferring user routes and locations using zero-permission mobile sensors. In *2016 IEEE Symposium on Security and Privacy (SP)*, May 2016.

[63] Sashank Narain, Amirali Sanatinia, and Guevara Noubir. Single-stroke language-agnostic keylogging using stereo-microphones and domain specific machine learning. In *Proceedings of the 2014 ACM Conference on Security and Privacy in Wireless & Mobile Networks*, WiSec '14, 2014.

[64] Sarfraz Nawaz and Cecilia Mascolo. Mining users' significant driving routes with low-power sensors. In *Proceedings of the 12th ACM Conference on Embedded Network Sensor Systems*, SenSys '14. ACM, 2014.

[65] Andrew Y. Ng and Michael I. Jordan. On discriminative vs. generative classifiers: A comparison of logistic regression and naive bayes. In *Proceedings of the 14th International Conference on Neural Information Processing Systems: Natural and Synthetic*, NIPS'01, Cambridge, MA, USA, 2001. MIT Press.

[66] US Department of Commerce. United States Census Bureau. https://www.census.gov/, 2018.

[67] OpenStreetMap. OpenStreetMap Amenity Key. http://wiki.openstreetmap.org/wiki/Key:amenity, 2017.

[68] OpenStreetMap. OpenStreetMap Building Key. http://wiki.openstreetmap.org/wiki/Key:building, 2017.

[69] Simon Oya, Carmela Troncoso, and Fernando Pérez-González. Back to the drawing board: Revisiting the design of optimal location privacy-preserving mechanisms. In *Proceedings of the 2017 ACM SIGSAC Conference on Computer and Communications Security*, CCS '17, 2017.

[70] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[71] A. Pingley, N. Zhang, X. Fu, H. A. Choi, S. Subramaniam, and W. Zhao. Protection of query privacy for continuous location based services. In *2011 Proceedings IEEE INFOCOM*, April 2011.

[72] R. Shokri, G. Theodorakopoulos, J. Y. Le Boudec, and J. P. Hubaux. Quantifying location privacy. In *2011 IEEE Symposium on Security and Privacy*, May 2011.

[73] Reza Shokri, George Theodorakopoulos, Carmela Troncoso, Jean-Pierre Hubaux, and Jean-Yves Le Boudec. Protecting location privacy: Optimal strategy against localization attacks. In *Proceedings of the 2012 ACM Conference on Computer and Communications Security*, CCS '12, 2012.

[74] Mingshen Sun, Tao Wei, and John C.S. Lui. Taintart: A practical multi-level information-flow tracking system for android runtime. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16, New York, NY, USA, 2016. ACM.

[75] Akiyoshi Suzuki, Mayu Iwata, Yuki Arase, Takahiro Hara, Xing Xie, and Shojiro Nishio. A user location anonymization method for location based services in a real environment. In *Proceedings of the 18th SIGSPATIAL International Conference on Advances in Geographic Information Systems*, GIS '10, 2010.

[76] Yufei Tao, Christos Faloutsos, Dimitris Papadias, and Bin Liu. Prediction and indexing of moving objects with unknown motion patterns. In *Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data*, SIGMOD '04, 2004.

[77] TeamWin. TeamWin - TWRP. https://twrp.me/about/, 2017.

[78] C. Tuduce and T. Gross. A mobility model based on wlan traces and its validation. In *Proceedings IEEE 24th Annual Joint Conference of the IEEE Computer and Communications Societies.*, March 2005.

[79] Tien Dang Vo-Huu, Triet Dang Vo-Huu, and Guevara Noubir. Fingerprinting wi-fi devices using software defined radios. In *Proceedings of the 9th ACM Conference on Security & Privacy in Wireless and Mobile Networks*, WiSec '16, pages 3–14, New York, NY, USA, 2016. ACM.

[80] Y. Wang, Dingbang Xu, Xiao He, Chao Zhang, Fan Li, and B. Xu. L2p2: Location-aware location privacy protection for location-based services. In *2012 Proceedings IEEE INFOCOM*, March 2012.

[81] Fengguo Wei, Sankardas Roy, Xinming Ou, and Robby. Amandroid: A precise and general inter-component data flow analysis framework for security vetting of android apps. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, CCS '14, New York, NY, USA, 2014. ACM.

[82] Ouri Wolfson and Huabei Yin. *Accuracy and Resource Consumption in Tracking and Location Prediction*. 2003.

[83] M. Xia, L. Gong, Y. Lyu, Z. Qi, and X. Liu. Effective real-time android application auditing. In *2015 IEEE Symposium on Security and Privacy*, May 2015.

[84] Xposed Framework. The Xposed Framework Source Code. https://github.com/rovo89/XposedInstaller, 2017.

[85] T. H. You, W. C.f Peng, and W. C. Lee. Protecting moving trajectories with dummies. In *2007 International Conference on Mobile Data Management*, May 2007.

[86] Hui Zang and Jean Bolot. Anonymization of location data does not work: A large-scale measurement study. In *Proceedings of the 17th Annual International Conference on Mobile Computing and Networking*, MobiCom '11, 2011.

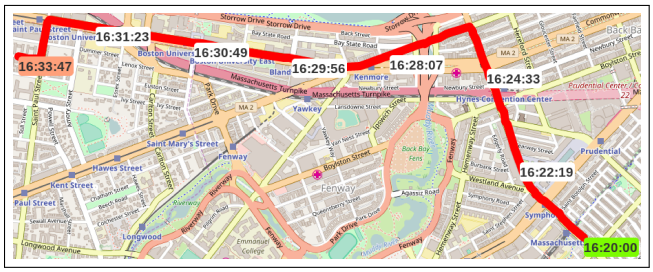
[87] L. Zhang, Z. Cai, and X. Wang. Fakemask: A novel privacy preserving approach for smartphones. *IEEE Transactions on Network and Service Management*, June 2016.

[88] Qunwei Zheng, Xiaoyan Hong, Jun Liu, David Cordes, and Wan Huang. Agenda driven mobility modelling. *Int. J. Ad Hoc Ubiquitous Comput.*, December 2010.

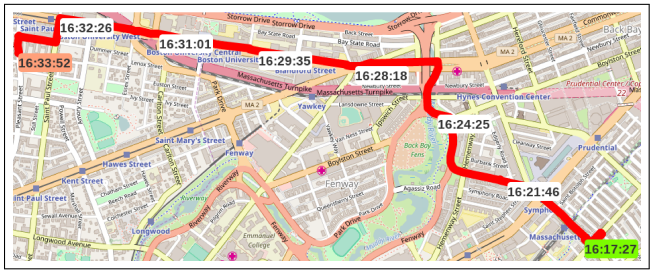
[89] J. Zhou, H. V. Leong, Q. Lu, and K. C. K. Lee. Optimizing update threshold for distance-based location tracking strategies in moving object environments. In *2007 IEEE International Symposium on a World of Wireless, Mobile and Multimedia Networks*, June 2007.

[90] Yajin Zhou, Xinwen Zhang, Xuxian Jiang, and Vincent W. Freeh. Taming information-stealing smartphone applications (on android). In *Proceedings of the 4th International Conference on Trust and Trustworthy Computing*, TRUST'11, Berlin, Heidelberg, 2011. Springer-Verlag.

[91] Suwen Zhu, Long Lu, and Kapil Singh. Case: Comprehensive application security enforcement on cots mobile devices. In *Proceedings of the 14th Annual International Conference on Mobile Systems, Applications, and Services*, MobiSys '16. ACM, 2016.



(a) Real Driving Route



(b) Generated Synthetic Trajectory

Fig. 8. An example of the similarity between a real route and a generated synthetic route.

Table 4. Cumulative results of the User Study on Mechanical Turk sorted by the number of noisy trajectories correctly labeled.

Noisy	Surveyors	Real Trajectories		Synthetic Trajectories	
		Real	Synthetic	Real	Synthetic
0	100	65.1%	34.9%	66.0%	34.0%
1	91	65.4%	34.6%	65.9%	34.1%
2	72	65.7%	34.3%	65.4%	34.6%
3	54	68.3%	31.7%	64.4%	35.6%

A Detection of Synthetic Trajectories by Regular Users

We also evaluated the realism of synthetic trajectories by conducting two separate user studies: one comprising of a group of 12 students from a university and another comprising of 100 users from Amazon Mechanical Turk [3]. We wanted to check whether it is possible for regular users to visually determine if the synthetic trajectories generated by MATRIX are fake. The intuition behind two studies was to understand the results from two perspectives; one of users who know the area very well and another of users unaware of the area. The university area was chosen so that the students were aware of its traffic congestions. The study asked the users to visually analyze a mix of 20 (10 real and 10 synthetic) trajectories and label them as 'Real' or 'Synthetic' based on their observations. Figures 8a and 8b show an example of a real route and a synthetic trajectory used for the study. The green marker marks the start location, the white markers are $500m$ apart, and the red marker marks the stop location. These markers display the time the vehicle was at the given location.

The trajectories were created as follows: First, we drove 10 unique routes close to the university area, each starting and ending at different locations and times of the day. Each route can be represented as $R = [n_1, \ldots, n_L]$, where n is a node and L is the number of nodes in the route. Each node n_i is attributed with timing and geographic information $n_i = (t_i, \text{Loc}(n_i))$, where t_i is the timestamp and $\text{Loc}(n_i)$ is the node's geographic coordinates. Next, we generated 10 synthetic routes using the timestamp of the first node (i.e., t_1) and geographic coordinates of the end nodes (i.e., $\text{Loc}(n_1)$ and $\text{Loc}(n_L)$) for each real route R. For 8 out of 10 routes, we observed that the real and synthetic trajectories were the same route, with slightly different timing information. For the remaining 2 routes, we observed that the real and synthetic trajectories were different but both were suggested routes by Google maps for the given source and destination. The trajectories were shuf-

fled so they appeared in a random order. For mechanical turk, we added three very noisy trajectories which looked obviously synthetic to find users who did not take the study seriously.

University Students Study: For the real trajectories, ≈ 64.2% of the trajectories were labeled as 'Real' and the rest were labeled as 'Synthetic'. For the synthetic trajectories, ≈ 65.8% of the trajectories were labeled as 'Real' and the rest were labeled as 'Synthetic'. Note that more users of this study confused the 'Synthetic' trajectories to be 'Real'.

Amazon Mechanical Turk Study: For the real trajectories, ≈ 68.3% of the trajectories were labeled as 'Real' and the rest were labeled as 'Synthetic'. For the synthetic trajectories, ≈ 64.4% of the trajectories were labeled as 'Real'. The above results are for 54 users who detected all the obviously noisy trajectories. Table 4 shows the cumulative results of the mechanical turk study based on the number of noisy trajectories detected by the users. We can see that the results are not significantly different even for all 100 users, however, more users labeled 'Synthetic' as 'Real'.

The results indicate that it was difficult for the users to differentiate between synthetic and real driving trajectories. There was confusion in both groups regarding their validity. Evaluating individual trajectories, we saw that this confusion applied to each trajectory as not a single one was labeled as 'Real' or 'Synthetic' unanimously by all users.

B MATRIX System Stability Evaluation

The system's stability was evaluated on 4 smartphones and the results are shown in Table 5. The evaluation was performed using 1000 popular apps on Google Play Store that requested location permissions or accessed the sensors. All the selected apps had a minimum rating of 4.0 and a minimum vote count of 10,000 users. These 1000 apps were successively run twice using an automated UI application exerciser tool called Android Monkey [7], once on a stock Android version of these smartphones and then with MATRIX installed on the same phones. The tool was configured to stress test each app's activities to monitor how many additional apps crash or fail to execute. The same settings were used for both tests ($seed = 1$, $num\_events = 2500$) to ensure that the same pseudo-random events were generated.

The first row for each phone in Table 5 shows the test results for the stock version and the second row shows the test results for MATRIX. All the apps installed and ran on every phone except for 15 apps on the HTC One M9 (possibly due to compatibility reasons). The number of successful monkey runs are very similar in both the tests with the stock version performing better on two phones and the MATRIX version performing better on the other two. We analyzed the errors/crashes manually to check for Xposed or MATRIX specific errors and did not find any. This validates that MATRIX remains stable and runs as expected for different devices, OS versions, apps and in heavy use.

Table 5. Results of the Stability evaluation for MATRIX using 1000 popular Android apps on 4 smartphones.

Phone	Version	Installed	Success	Failure
HTC One M7	Lollipop	1000	892	108
		1000	894	106
HTC One M9	Marshmallow	985	796	189
		985	791	194
LG Nexus 5	Lollipop	1000	938	62
		1000	944	56
LG Nexus 5X	Marshmallow	1000	851	149
		1000	848	152

C PrivoScope User Study: Questions and Responses

Below are the questions asked to the users of the study. All response values ranged between 0 (no knowledge / not useful) to 5 (very knowledgeable / very useful).

- **Q1:** How knowledgeable were you about location tracking by the GPS and WiFi before this demo?
- **Q2:** How knowledgeable were you about location tracking by the sensors (Accelerometers, Gyroscope and Magnetometers) before this demo?
- **Q3:** Rate your level of satisfaction with the protections that current mobile Operating Systems provide to protect your location information.
- **Q4:** Based on the demo, how useful do you think PrivoScope is for understanding how installed apps on your device access your private information?
- **Q5:** How likely are you to use PrivoScope, and recommend it to your friends and family?
- **Q6:** Based on your usage of the device, did you notice any performance degradation with PrivoScope active on the device? (5 means no degradation)

– **Q7:** Based on your observations about the flashlight apps, which app would be keep while removing the others? (App1 / App2 / App3)

Table 6 provides a summary of the responses from the users of the study.

Table 6. Responses from the PrivoScope User Study

User	Q1	Q2	Q3	Q4	Q5	Q6	Q7
User 1	5	3	1	5	5	5	App1
User 2	4	4	2	4	5	5	App1
User 3	5	4	5	5	5	5	App1
User 4	2	2	2	5	5	5	App1
User 5	4	3	3	5	5	5	App1
User 6	2	1	3	4	4	5	App1
User 7	2	1	2	5	5	5	App1
User 8	4	2	3	5	5	5	App1
User 9	4	3	3	5	5	5	App1
User 10	2	1	3	5	5	5	App1
User 11	2	2	3	5	5	5	App1
User 12	3	3	2	5	5	4	App1
User 13	4	3	4	5	5	5	App1
User 14	4	3	2	4	4	5	App1
User 15	3	2	4	5	5	5	App1