Selective Cloaking: Need-to-know for Location-based Apps

Benjamin Henne, Christian Kater, Matthew Smith, Michael Brenner
Distributed Computing & Security Group
Leibniz Universität Hannover
Germany
Email: {henne, kater, smith, brenner}@dcsec.uni-hannover.de

***Abstract*—Currently, location privacy settings of mobile operating systems are limited to the option of enabling or disabling the use of location completely or on a per-app basis. When location use is allowed, users always reveal their location in full precision even to apps that do not need it. For instance, weather forecast apps and navigation apps both get the most exact location a device can determine. Up to now, mobile privacy research was focused on the recognition and prevention of disclosure of private data. This includes location data, but does not extend to privacy in use cases where users do want to disclose their location — but not in full detail. However, the increasing adoption of smartphones entails the increasing use of location-based services as well. Users want to use these services, but have privacy concerns. As many location-based services do not require exact locations, user privacy can be increased by only disclosing location in such detail as required for the respective service to function. To enable users to restrict the accuracy of location data that is revealed to apps, we created a location privacy framework that allows per-app location obfuscation. The framework allows easy integration of different obfuscation algorithms into the Android system. We present both on-device obfuscation and service-based obfuscation and evaluate our framework.**

I. Introduction

Since the beginning of the smartphone era, the number of mobile apps that use location data has increased rapidly. The prominent mobile operating systems used on smartphones, tablets and modern digital cameras integrate location services, based on GPS, Wi-Fi or cell tower tracking. Mobile apps can gain permission to use location data for different types of location-based services ranging from local weather reporting to check-ins at specific places like coffee shops to notify friends about one's whereabouts. All current operating systems allow users to enable or disable location services for all apps on a device at once, but not all allow selective per-app configuration that allows users to preserve their location privacy while not abdicating location use at all. While Apple iOS allows selective disabling since version 6.0, Android as the mobile OS with the highest market share does not integrate this indispensable privacy feature up to current version 4.2 Jelly Bean.

Android users can mend this issue by using special apps or custom ROMs that allow them to restrict location permissions on a per-app basis. However, only a subset of all Android devices allows this. In the first case, the installed OS must be supported by such an app that in turn requires the user to have root privileges. In the second case, the device must be able to be flashed with a custom ROM. Even if users could restrict the use of location data by customizing permissions of an app using a ROM like CyanogenMod 7 [1], some apps simply crashed, as they assume that they have full permissions. In addition to technical reasons, many users are not skilled to use either of these workarounds or are not willing to loose warranty.

Existing location privacy features only allow for a limited preservation of location privacy: Depending on the OS features, users can disable location use completely or for specific apps. However, it is not possible to control the level of accuracy of location information used by their apps. Maps or routing apps, location sharing apps, local weather forecast apps as well as music services, all get access to the same location data, independently of the use case or the needed location accuracy. Whenever users want to use a location-based service, they must reveal their exact positions to an app even if the app does not need it in that detail. On Android, the differentiation between permissions for coarse and precise location allows for accuracy control to a limited extent, but the selection of permissions is still made by app developers, not by users. iOS currently lacks any such differentiation. While some mobile OSs lack per-app location use on/off switches, none of them allows users to control location accuracy.

In contrast, there have been numerous works on the modification (i. e. obfuscation) of location data in privacy research that aim at preserving user privacy. However, most location obfuscation research is only of theoretical nature, neither evaluating the real world applicability on mobile devices, nor the usability aspects for users. Research suggests very different solutions ranging from single-user on-device obfuscation algorithms that basically change coordinates and accuracy within given ranges [2] to multi-user Web-based services that guarantee non-distinguishability within a given number of users [3]. Such location obfuscation algorithms are still missing in mobile OSs. They would allow users to control location exactness to achieve the best possible privacy enhancement based on personal requirements.

Besides works on location obfuscation, there are also different studies on user feelings about disclosing location data [4], [5]. Some more recent publications state that location data still raises concerns for today's users [6] and that especially on mobile devices other data might raise more privacy concerns [7]. However, users still seem to pay attention to location privacy [8] and they are becoming aware of which kind of apps they allow the use of location data [9].

Especially on mobile devices users may use location data in different scenarios such as geo-tagging photos, using location-based services or location sharing, which may also have different privacy requirements [10]. Thus, location privacy and its preservation, especially in the mobile ecosystem, require further research. In this paper we argue that more than a per-app location on/off switch is needed to offer users the best privacy/functionality trade-off. Users should be enabled to obfuscate their location on a per-app basis to give each app only the minimum amount of location information needed to operate.

In this paper we present a location privacy framework for Android that implements location obfuscation on a per-app basis. The framework allows developers to easily implement different location obfuscation algorithms on an Android device. Users of the modified Android system are able to select and configure different obfuscation algorithms for each installed app that requests location data. In this work, we show how location obfuscation can be integrated into current mobile devices. We implemented and evaluated the framework and exemplary obfuscation algorithms and present results and limitations. In Section II, we present the design of the location privacy framework and its integration into the Android system. Section III presents the obfuscation algorithms that have exemplarily been implemented with the framework. In Section IV, we sketch a first evaluation of the framework and discuss some usability aspects of location obfuscation algorithms in general in Section V. We present related work to this paper in Section VI. Finally, we conclude our work and present possibilities for future research in Section VII.

II. Location Privacy Framework

The location privacy framework allows Android users location obfuscation for all apps that use location data. Basically, obfuscation can be switched on and off for the whole device. Location data requested by an app is obfuscated using a default obfuscation algorithm if nothing app-specific was configured. Obfuscation is handled as opt-out feature for newly installed apps. Users can enable or disable obfuscation for single apps. Additionally, they can assign each app a different obfuscation algorithm with different parameter values. The configuration of the framework is done via the Android Settings app. The framework has been implemented[1] as an extension of the custom ROM CyanogenMod 9.1 (Android 4.0.4 Ice Cream Sandwich) [1] on a Samsung Galaxy Nexus smartphone.

The framework consists of three components: First, the model of location privacy algorithms and configurations; second, the control unit that applies configured privacy algorithms to location requests; third, the extension of the Settings app that allows users to configure privacy algorithm settings for each app requesting location data.

A. Framework Model

The framework model provides the *AbstractLocationPrivacyAlgorithm* as a template for the implementation of obfuscation algorithms. For each obfuscation algorithm, the following methods have to be implemented: *obfuscate()* is called by the framework to obfuscate a location. It is provided with the original location data and has to return the obfuscated one. The provided Context allows more complex implementations the use of other services, such as Android's built-in features like the Geocoder API or a service on the Web.

The method *getDefaultConfiguration()* is used to define configuration parameters and default values of the algorithm's *LocationPrivacyConfiguration*. If the algorithm is applied to an app, these default values are used and can be altered in the UI. The visibility of the different parameters in the Settings app is also defined in the LocationPrivacyConfiguration: Parameters can be visible and alterable by users, alterable but masked like passwords, or hidden. The defined type of a parameter (like integer, boolean or coordinate) also influences the visualization and modification Activity used in the Settings app. To allow the configuration of an algorithm in Settings, the XML-based UI definition and localization has also to be extended.

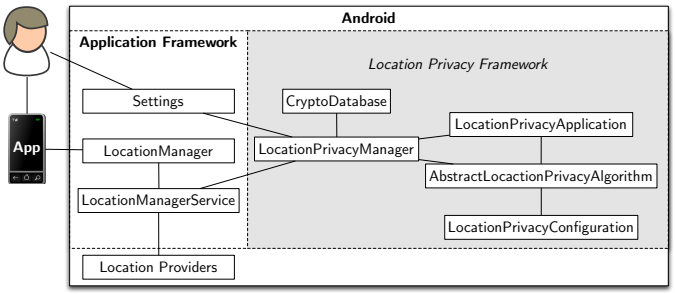

Fig. 1. Location Privacy Framework Components and Integration

B. Framework Control

The *LocationPrivacyManager* (cf. Figure 1) provides all actions for the management of the framework and the obfuscation of locations requested by apps.

All configuration data of the framework is stored in the encrypted *CryptoDatabase* that is based on SQLite. SQLite is included in Android by default, but the default library does not allow the encryption of data. To keep our implementation lightweight, we did not exchange the default library by one that also implements encryption. Instead, the framework includes the CryptoDatabase, which is a wrapper around SQLite implementing the missing encryption. The CryptoDatabase implements all SQL command except ORDER BY. If a command is executed, it encrypts data before writing or comparing to the database and decrypt data that is read from it. AES is used for data encryption.

When the LocationPrivacyManager is created on startup, it loads the secret for database encryption from private SharedPreferences. If no secret exists on the first instantiation, a random secret is generated. The LocationPrivacyManager instantiates the CryptoDatabase for accessing the configuration from its SQLite database. All configurations are buffered in memory to minimize database file access. If the user modifies the framework configuration in the Settings app, the LocationPrivacyManager reloads configuration data from the file system triggered by a broadcast received via the LocationManagerService.

The configuration database stores the general configuration of the framework, such as its default obfuscation algorithm,

[1] Source: http://bhenne.github.io/android-location-privacy/

```
Data: real location, uid of app, name of app
Result: location
  if real location is not null then
    location ← real location
    if framework is enabled then
      if uid is in configBuffer then
        app ← bufferedConfig(uid)
      else
        app ← configFromDB(uid)
      end if
      if app still is null then
        app ← registerApp(uid, name)
      end if
      if app obfuscation is enabled then
        algorithm ← app.getAlgorithm()
        algorithm.setContext(frameworkContext)
        location ← algorithm.obfuscate(real location)
      end if
    end if
    return location
  end if
  return null
```

Fig. 2. Location obfuscation in LocationPrivacyManager

whether or not it is enabled and parameter values. For each registered app, the database stores if obfuscation is enabled for the app, its obfuscation algorithm and parameter values.

The method *obfuscateLocation()* implements the obfuscation of a location (cf. Figure 2): If the method was called with the *null* location, the *null* location is returned. If the framework is completely disabled, the unaltered location is returned. Otherwise the framework first looks for the app in its configuration buffer using the app's *uid* assigned at installation time. If no matching configuration is buffered, the framework makes a lookup in the database. If the configuration for the app is neither buffered nor in the database, the app is using location data for the first time. In this case the app is entered into the framework using the default obfuscation algorithm. Finally, if obfuscation is not disabled for the app, the location is obfuscated using the configured algorithm.

C. Settings App Extension

The extended Settings app allows users the configuration of the location privacy framework, i. e. the used obfuscation algorithms and parameter values. The configuration has been integrated with the other security and privacy settings as shown in Figure 3. The switch in the Settings main view gives users a quick indicator to see if obfuscation is currently enabled or not. The location privacy settings (cf. Figure 4) show which apps already requested location data, which algorithm currently is set for each app, and if obfuscation is enabled.

By selecting the settings of an app, users can setup detailed configuration. They can select an algorithm from the list of available implementations. For each algorithm, parameters can be specified according to the model. Hidden parameters (e. g. last location for location change constraints) are not shown and secret values like a password are masked with dots. Depending on the type of a parameter an appropriate soft keyboard is used. Boolean values are visualized as check boxes and lists

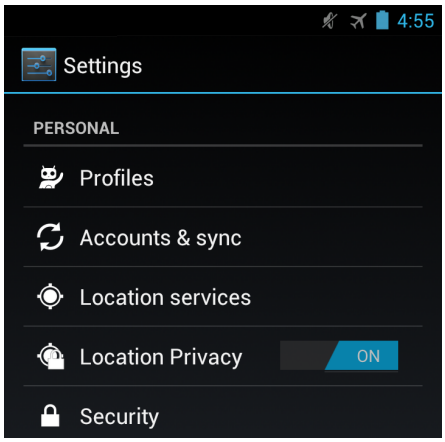

Fig. 3. Android Settings

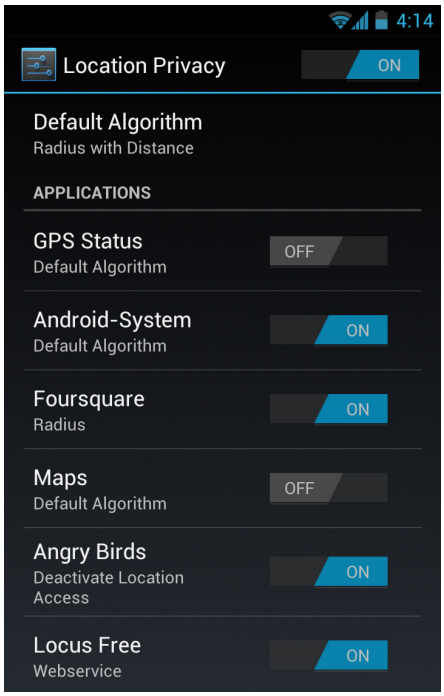

Fig. 4. Android Location Privacy Settings

are visualized as drop-down selections. If a user selects a coordinate parameter, a map Activity is opened: The user can select a coordinate on a map, can search for an address or can simply enter coordinates values. Android's XML-based UI definition is used to specify the existing algorithms' texts: The algorithms' names, their parameters and parameter descriptions can be set up localized in different languages.

D. OS Integration

An Android app can access the current location if location services are enabled and the app requested as well as was granted the permissions for location data during installation. To access location information, an app creates an instance of *LocationManager*. Each LocationManager instance gets location data from the singleton *LocationManagerService* that is running in the OS context in Android's application framework

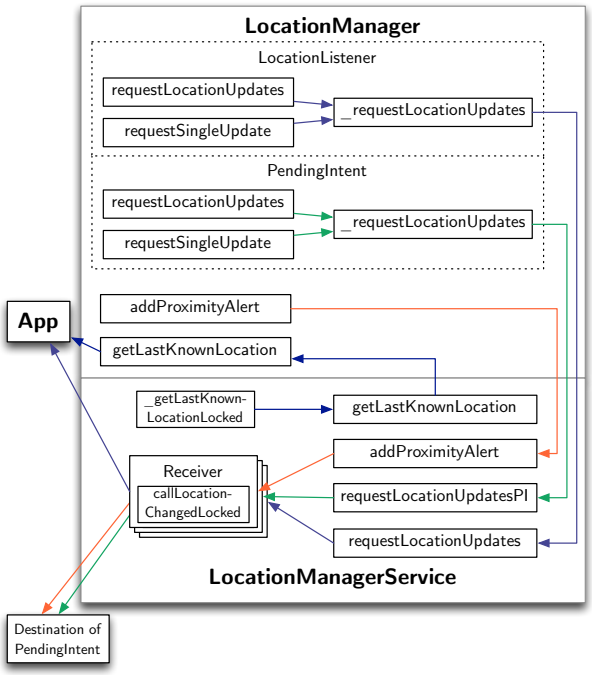

Fig. 5. Flow of location information in Android Ice Cream Sandwich (ICS)

layer. The LocationManagerService itself receives location data from different location providers in the library layer, such as the one for GPS (cf. Figure 1).

On Android 4 ICS, apps have different ways for acquiring location data: An app can access the last known location of a device by calling the method *getLastKnownLocation()*. Additionally, apps can register *LocationListener*s to get up-to-date location information. Based on conditions such as the minimum distance or time between two locations, the listener is updated with new location data. Similar to location listeners, apps can also register a *PendingIntent* to acquire updates about the current location. They also can register a *ProximityAlert* for a given location. All location data requested by an app via a LocationManager passes the *LocationManagerService* as shown in Figure 5: Every location passes its *getLastKnownLocation()* or *callLocationChangedLocked()* method. These are the integration points of the location privacy framework at which location data is obfuscated before it is returned.

The LocationManagerService is running in an own process separate from user apps. It needs to know which obfuscation algorithm to use if location data is requested. The attribution of a location request to an app and its configured obfuscation algorithm is done using the app's uid. Android provides this information using *getCallingUid()* of the *Binder*. When an app is installed on the Android system, a uid is assigned to the app and subsequently used for running the app. This uid may not always be unique in the system: Some apps may run with a shared user id. If multiple apps use the same uid, the name of the app that requested location data first using a uid is used in the configuration database. All location requests by that uid are handled using this configuration. The names of other apps using the same uid may be added for transparency.

Users do not need to add apps to the framework manually, nor has the installation routine of the system been modified. Our solution is less invasive: It discovers new apps when they request location data for the first time. Hence, less components of the system needed to be extended. We chose location privacy to be enabled by default, since privacy always should be an opt-out feature. In this case the framework does not require its users to actively think about privacy each time they install an app. Only if they need original location data or non-default obfuscation, they have to change settings. If an app is removed from the system, its location privacy configuration is removed when the location privacy settings are opened the next time.

The LocationManagerService is running within the Context of the Android system, which is the same Context that the Settings app uses. The shared context allows the shared access and storage of configuration between both parts of the framework and no inter-process communication is required.

E. Threat Model

The location privacy framework is running in the OS's context. Unprivileged apps neither can access the framework's memory, nor can they maliciously modify the configuration of the framework, because the configuration stored in the file system is encrypted with the secret stored in the inaccessible SharedPreferences. By this, no unprivileged app can modify or disable obfuscation that has been set up by the user. Only apps with root privileges can bypass obfuscation by manipulating the location privacy framework.

The location privacy framework was mainly built to support users at preserving their privacy when using location-based services or disclosing their locations. These requirements are fulfilled. The framework does not protect users from malicious tracking attacks. In such cases, attackers can use much simpler attacks to circumvent privacy measures: Leveraging cell tower ids, Wi-Fi SSIDs and MAC addresses, an attacker can locate a user using corresponding external location services. Malicious apps thus still need to be discovered and banned by the app stores. A non-malicious example of bypassing our privacy framework was observed in one trustworthy app within our evaluation: If we disabled GPS, the Google Maps app received real location data even if all active location providers returned obfuscated locations via the LocationManagerService (cf. Section IV-B).

On the other hand, it is conceivable that an attacker abuses the location privacy framework in order to spoof location data. However, there already are other apps for example using Android's mock location provider, which allow this kind of spoofing. Hence, the location privacy framework does not introduce new avenues for malicious activities.

III. Obfuscation Algorithms

We implemented the following set of obfuscation algorithms in the current version of the framework.

A. On-Device Obfuscation

On-device obfuscation algorithms need no access to any service on the Internet. They even work if users have no active network connection. These algorithms are faster, because they

are not slowed down by any network delay. They also are of simpler nature and are satisfactory for many use cases. These algorithms do never reveal real location to any external privacy service. The framework includes the following four on-device algorithms:

Deactivation of location data implements per-app disabling for Android as implemented by iOS 6. By simply returning the *null* location, an app does not get location data, keeps working, and does not fail as in the case of revoked permissions.

Fixed coordinates allow users to set up fixed coordinates for an app. A user can simply pick a location on a map that will never change for the app.

Random shift with maximum distance selects random coordinates within a circle with given radius [2]. The user specifies the radius in meters as well as a location change constraint distance: The random location is only changed if the user moved this distance in reality. This avoids that random location changes at each request draw a scatter-plot on the map, which would make deriving the user's real location easy. It also prevents a user's location from continuously jumping on the map, which might negatively affect location use in different use cases.

Random shift with minimum and maximum distance extends the previous algorithm. By the definition of a minimum distance constraint users prevent random coordinates to be very near or even equal to their real location, which can happen, but with low probability.

B. Service-Based Obfuscation

Service-based obfuscation algorithms use external services and hence require a working Internet connection. Yet, most use cases involving location information also require an Internet connection, for example when requesting weather data or searching for a nice café nearby. These algorithms are slower due to network delay, but they also allow more complex obfuscation relying on external parameters, such as the location of other users of an obfuscation service. Using such algorithms, users share their real locations with a privacy service or spatial data service, while hiding it to others. Prominent examples of such algorithms are those achieving *k-anonymity* [3]. We implemented two service-based algorithms:

Web service simply allows the use of an external obfuscation service. This algorithm is configured with a secure web service address, username and password. Using this algorithm, it calls the web service with real location data, authenticated using HTTP basic auth, catches the reply assumed to be formatted in JSON and returns the location returned by the service to the app. This algorithm for instance could be used to call an obfuscation service that depends on multiple users, such as a k-anonymization service.

Geo-coding is an algorithm we propose that relies on the combination of reverse geo-coding and geo-coding. When a location is requested, the original coordinates are reverse geo-coded to get its address (street, district, city, region, state and country). Based on the privacy detail level configured by the user, parts of the address information are removed to lower its accuracy. Then the modified address is geo-coded again. The resulting coordinates are returned to the requesting app. Therefore, the algorithm maps the location of a user to the center of the bounding box of a geographic datum, which might be a street, a city boundary or frontier. One drawback of this algorithm is that it relies in the data quality of the underlying spatial data service. However, different free services, such as those based on OpenStreetMap, or services served by Google are sufficiently accurate for this use case. The positive aspect of this algorithm is that it might be easy to grasp for users, because they do not have to specify any numbers, but only have to choose a detail level saying "*I am in this street*" or "*I am in this city*". Currently the levels *street*, *postal code*, *city* and *country* are implemented. People, which are near each other, that chose the same level of detail are mapped to the same coordinates. This might be beneficial for some use cases, but needs to be researched further. The geo-coding algorithm currently uses the Geocoder API integrated in Android and hence relies on Google's service as a third party service that is used twice for each obfuscation. One could implement this algorithm independently from Google by using another geo-coder, like the free Nominatim by OpenStreetMap or OpenMapQuest, or by implementing the whole algorithm as an independent service based on those open-source technology and data. It would also be possible to implement a two-layer onion routing protocol for further privacy.

IV. Framework Evaluation

In the following, we present a basic performance evaluation of the implemented location privacy framework. We also identified some limitations and discuss usability.

A. Performance

The use of the location privacy framework should not have a large impact on system performance. We ran tests using a sample app that poses multiple requests to *getLastKnownLocation()*. When calling this method, no time is used for any positioning via a LocationProvider, because a buffered location is used and obfuscated. The timing tests were made on a freshly flashed Samsung Galaxy Nexus with a dual-core ARM Cortex-A9 processor and 1GB RAM.

We made timing tests with respect to four setups: (1) CyanogenMod 9.1 without the location privacy framework; (2) the extended OS with disabled framework as well as the framework enabled and obfuscation algorithm of the test app (3) disabled and (4) enabled. For each setup we measured multiple runs, within each posing 10,000 location requests. The timings were measured using an on-device algorithm: random shift within the maximum radius. We could not measure any noteworthy impact: Independent of the setup each run of 10,000 calculations took between 4.2 seconds and 4.9 seconds. Thus, we conclude that the framework itself does not slow down the system in a measurable way.

We also compared the implemented algorithms. The device was connected to the Internet via our university network Wi-Fi access point for these tests. A test web service was located within the local network. Table I shows the results of the test runs. The maximum of the on-device obfuscation algorithms was caused by changing the used algorithm within tests: After changing the algorithm, the configuration is reloaded at the next request causing that request being relatively slow. The

TABLE I. DURATION OF LOCATION QUERIES

obfuscation algorithm	time [seconds]		
	min	avg	max
fixed	0.0008	0.0012	0.0019
random (max)	0.001	0.0013	0.005
random (min/max)	0.001	0.0015	0.005
web service	0.2	0.23	0.4
geo-coding	0.2	0.46	1

average time of the on-device algorithms can be regarded as equal. Their computational load is equally low. The high results of the geo-coding algorithm are caused by the response times of the two calls to the Android Geocoder API, which are higher than those of the request sent to web service in the local network. Overall we believe the obfuscation time is tolerable. In the case of the geo-coding algorithms, users might notice a small delay, if location requests are not made in the background.

B. Limitations

While testing different use cases and apps that all worked properly with the location privacy framework, we found Google Maps (including Google Local) does not work entirely as expected: If GPS was up and running, the Maps app used the obfuscated location as expected. However, if GPS was deactivated or no fix could be established, the Maps app showed the real location – in these cases the location was based on cell towers or the location of Wi-Fi access points. It seemed as if the Google Maps app uses its own location service to locate the device based on Wi-Fi and cell tower tracking instead of using Android's built-in network location provider, if GPS is not available. Or, the app might be able to bypass the LocationManagerService and hence directly accessing the NetworkLocation provider in some way. Any call to *getLastKnownLocation("network")* returned no cached location but *null* after using Maps on a freshly booted device. Using any other app requesting location from the network location provider proved that network-based location was obfuscated correctly. Using a HTTPS proxy, we found Maps using a web service at `google.com/glm/mmap` that is known as a web-based location service. Transmitted data was in binary format and we only could recognize our current access point's SSID and MAC address. Therefore, we were unable to identify if the request contained a location query or only anonymous location data sent to the service. Finally, we could not completely prove either of the two guesses. It is interesting to note, that when we disabled Google's location service (normally disabling the built-in network location provider) in Settings, Maps announces that is not able to get any location.

Irrespective of the way Google Maps gets real location information when GPS is not available, whenever an app implements its own location service, the OS's location service is circumvented and hence the location privacy framework cannot obfuscate the determined location. This is a general problem that cannot be solved externally. The location privacy framework's algorithms are only applied to location data if the location service of the Android system is used. Google Maps was the only app we found while testing that circumvented the location privacy framework. Other apps as for instance Camera, GPS Status, Locus Map, Foursquare, Yelp, the default browser and Google Chrome getting a Javascript location request as well as an app we created for testing the Google Maps API worked as expected.

C. Usability

The presented location privacy framework can be used as is. At the same time, technically skilled or privacy concerned users can adapt the settings to their needs. However, by the definition of reasonable default values and a reasonable default algorithm the framework provides increased location privacy for all users. The customizing of parameters and the selection of different algorithms for different apps and use cases still needs basic knowledge though. Enabling even unskilled users to make suitable detailed configurations should be the goal of future work as discussed in the next section.

V. USABLE LOCATION OBFUSCATION

Regarding real world applications, usability of location obfuscation algorithms plays an integral role. The main problem with the multitude of location privacy preserving techniques is that most might not be easy to understand for users and that algorithms have different parameters that have to be set up by someone. If we require users to specify such parameters, most of them will not be able to define appropriate parameters according to their personal privacy preferences, because many parameters are hard to grasp or understand. For instance, if using a basic k-anonymity algorithm, what does it mean for users if they are not distinguishable within a group of 50 people relating to location accuracy? How does it affect the location obfuscation if the user is in a large city like New York City or somewhere in the Australian Outback? How much location inaccuracy is actually created? Even if users might be able to reflect on these differences, most would not be able to answer such questions or set a k to match their privacy wishes. Even much simpler algorithms might be hard to comprehend. For instance, what privacy benefit is achieved by a random location shift within a distance between 50m and 500m? Does this have different consequences if being in a big city or in the countryside? And lastly, how will users decide which kind of location accuracy is adequate for different services or use cases.

While privacy research has started to compare different obfuscation algorithms [11], to date there have been no user studies evaluating different algorithms and parameters in real world settings. We plan to fill this gap in future work.

The presented location privacy framework currently does not address these usability issues. However, it builds a basis for the real life evaluation of obfuscation algorithms. The location privacy framework allows the implementation of different algorithms on real devices. This allows researchers to conduct user studies in real world scenarios. Participants can use obfuscation, and their experience can be assessed. This allows algorithms to be evaluated from the users' perspective and can uncover real world problems of parameter setting. Based on studies of user experience, a usable location privacy framework would include:

- Applicable algorithms have to be selected from the multitude of research proposals. The goal is finding

those algorithms that can be used in real world scenarios: While users require location privacy, many might not need the most effective algorithm, but rather a solution that allows a certain level of privacy [8].

- Selected algorithms must have user comprehensible differences concerning parameters or resulting privacy.
- Research has to identify concepts that allow users to grasp what they have to configure. For instance, while many people might feel uncomfortable with entering distances as sheer numbers, maybe metaphors like a soccer field might be clearer. Or, when specifying a value k for k-anonymity, people may prefer defining being as private as in a shopping mall for instance [12]. Some algorithms already include more easily graspable parameters by design: For instance, the geocoding algorithm we propose in this work uses known concepts like street, district or city.
- Evaluating common settings of particular user groups might lead to pre-defined parameter sets for different kinds of privacy awareness, for instance ranging from privacy addict to post privacy, or maybe wizard-based configuration may help setting up algorithms according to users' personal needs.

VI. Related Work

Numerous location obfuscation algorithms have been proposed in the last years [6]. They can be divided into two groups corresponding to our implementation in Section III:

First, algorithms running on the mobile devices. In [2], Ardagna et al. present different obfuscation operators that solely rely on geometric transformation of coordinates. These algorithms can completely be implemented on a mobile device as exemplarily implemented by our framework.

Second, algorithms using some middleware to integrate external anonymization services. The web service client algorithm we implemented in the framework addresses centralized trusted third party (TTP) location anonymization services. Gedik and Liu [3] present a corresponding k-anonymity approach: Users query the TTP service with their current location and a value k, and the service returns the location of a cloaking box that spans the location of at least k service users. By using this cloaking box as location data, a user is not distinguishable out of k people. Besides TTP services, also different decentralized cooperative services have been proposed. For instance, Takabi et al. [13] present a collaborative approach for achieving k-anonymity.

Different approaches to location privacy are hard to compare with respect to the achieved level of privacy. Shokri et al. [11] present a framework that aims at quantifying location obfuscation algorithms in the light of different attacks. The quantification allows comparing different obfuscation algorithms. The developed *Location Privacy Meter* can be used by designers to evaluate obfuscation algorithms.

In [14], Benisch et al. present a survey of 27 subjects on location privacy preferences and accuracy of policy rules' effects. They examine different privacy policies specifying which people (family, Facebook friends, advertisers, etc.) may view users' locations. This policy driven approach does not use obfuscation, but allows or disallows location per-user access. Our framework aims at applying a first obfuscation before sharing any location to people and service providers. Additional policies may be applied in a second step. However, some policy dimension studied in this work like time of day or blacklisted locations may be possible parameters for automated adjustment of inaccuracy created by obfuscation.

Brush et al. present in [15] a first study of 32 participants on long-term location tracking including participants' feelings about the application of obfuscation algorithms. After 8 weeks of recording GPS data, participants were asked in an interview, using a personalized questionnaire showing their location data, to which audience they would share their personal GPS track if applying each of 5 different obfuscation methods in 4 different obfuscation levels. Participants were most comfortable with applying k-anonymity, followed by deleting location near private places and adding Gaussian noise in contrast to discretizing coordinates and subsampling of the timescale. Participants that were guided through the questionnaire seemed to have understood basic operations of these methods, but seemed still not having understood implications of the data and obfuscation levels. This work focuses on the sharing of location tracks and user preferences for obfuscation algorithms. It does not address their usability aspects.

No current mobile OS integrates location obfuscation algorithms. All of them allow enabling or disabling the use of users' current location, but differ in the granularity of privacy control. Apple *iOS* allows its users to enable and disable the use of location on a per-app basis since iOS version 6. In previous versions, a single switch was used to enable/disable location use for all apps at once. *Android* only offers a single switch for location use for all apps installed on a device. Some customizations added more location privacy features in the past. For instance, *CyanogenMod 7* [1] implemented additional application permissions controls that allow users revoking permissions for apps that requested those at installation. In the case of location permissions, the solution worked well in some cases, but also stopped working some apps that crashed because they could not handle missing permissions. The feature was removed again in CyanogenMod 9 when switching to the Android 4 code base.

Some privacy-focused Android apps allow users the restriction of private data. These require OS root privileges. *LBE Privacy Guard* [16] runs in the background and monitors app activities. If an app requests private data like location, the user is alerted and asked whether to deny or allow access for that single request. Alternatively, users can completely allow or deny access to that private data for all apps in its settings. *PDroid 2.0* [17] has similar features. It allows blocking access to private data. For location data is also allows setting a fixed fake value.

Research proposed different extension of the Android system that allow better tracking and protection of personal data. *TaintDroid* [18] is a taint tracking and analysis system that allows tracking and visualizing of what private information of users is used and sent by apps. *AppFence* [19] goes a step further and allows the substitution of shadow data in place of data users want to keep private. It additionally allows blocking network transmissions of such data. Location is one of the

private data that is protected by AppFence. If location data is shadowed, it is replaced by a fixed location. AppFence aims at not releasing real location to services and advertisers that do not need this information for the real purpose of an app. It is not in conflict with our location privacy framework that aims at optimizing location privacy by modifying the exactness of location. In the case of location-based services, AppFence rather breaks functionality than improving privacy. It is applicable for securing location data against advertisers, but someone in Great Britain may not be interested in the local weather at the Googleplex in Mountain View, CA. *MyShield* [20] instead does basic reducing of location exactness: In the case of trusted apps, it simply rounds coordinates to one decimal place. Untrusted apps are neglected to receive location data while the authors noted other solution as completely random coordinates or using (0, 0), which is somewhere in the Gulf of Guinea, Africa. This is also not in conflict with our solution, since it does not implement real obfuscation algorithms. While our solution allows users to control app-specific algorithms with different accuracy, they only apply some general rounding that has a partially unpredictable effect depending on coordinate decimal places.

VII. Conclusion and Future Work

In this work, we presented a location privacy framework for Android that allows the easy integration of different location obfuscation algorithms into Android's location services. The location privacy framework enables users to disclose location data with different inaccuracies specified per-app depending on location use cases: Weather apps work properly with city-level location data; users would maybe rather choose street-level locations when sharing with friends; most add-funded games may never need location information, while in contrast, navigation software needs exact location data.

While many other works focus on the recognition and restriction of location disclosure, our framework aims at privacy-aware active location use within the diversity of today's location-based services. It preserves users' location privacy by minimizing the level of information that is disclosed while keeping services working. The location privacy framework is focused on the obfuscation of location information. It will not detect location misuse. However, it could be combined with other solutions presented in Section VI to achieve both goals.

Currently, the framework needs users to set up a location privacy configuration if an app needs non-default obfuscation. Users have to select algorithms and specify parameters, which might be a burden for unskilled users. At this point, usability research is needed in future work as argued in Section V. Additionally, users should be supported in the process of specifying which apps need or should use what location accuracy to achieve the best tradeoff between location-based service usage and location disclosure.

References

[1] Cyanogenmod. [Online]. Available: http://www.cyanogenmod.org/

[2] C. Ardagna, M. Cremonini, S. De Capitani di Vimercati, and P. Samarati, "An obfuscation-based approach for protecting location privacy," *Dependable and Secure Computing, IEEE Transactions on*, vol. 8, no. 1, pp. 13–27, 2009.

[3] B. Gedik and L. Liu, "Protecting location privacy with personalized k-anonymity: Architecture and algorithms," *IEEE Transactions on Mobile Computing*, vol. 7, no. 1, pp. 1–18, Jan. 2008.

[4] S. Consolvo, I. E. Smith, T. Matthews, A. LaMarca, J. Tabert, and P. Powledge, "Location disclosure to social relations: why, when, & what people want to share," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ser. CHI '05, 2005, pp. 81–90.

[5] S. Ahern, D. Eckles, N. S. Good, S. King, M. Naaman, and R. Nair, "Over-exposed?: privacy patterns and considerations in online and mobile photo sharing," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ser. CHI '07, 2007, pp. 357–366.

[6] J. Krumm, "A Survey Of Computational Location Privacy," *Personal and Ubiquitous Computing*, vol. 13, no. 6, pp. 391 – 399, 2009.

[7] A. Porter Felt, S. Egelman, and D. Wagner, "I've Got 99 Problems, But Vibration Ain't One: A Survey Of Smartphone Users' Concerns," in *Proceedings of the 2nd ACM workshop on Security and privacy in smartphones and mobile devices*, ser. SPSM '12, 2012, pp. 33–44.

[8] B. Henne and M. Smith, "Awareness about photos on the web and if privacy-privacy-tradeoffs may help," in *Financial Cryptography and Data Security (FC13), Workshop on Usable Security (USEC13)*, 2013.

[9] D. Fisher, L. Dorner, and D. Wagner, "Location Privacy: User Behavior In The Field," in *Proceedings of the 2nd ACM workshop on Security and privacy in smartphones and mobile devices*, ser. SPSM '12, 2012, pp. 51–56.

[10] B. Henne, M. Harbach, and M. Smith, "Location privacy revisited: Factors of privacy decisions," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems, CHI'13, Extended Abstracts*, 2013.

[11] R. Shokri, G. Theodorakopoulos, J.-Y. Le Boudec, and J.-P. Hubaux, "Quantifying location privacy," in *Proceedings of the 2011 IEEE Symposium on Security and Privacy*, ser. SP '11, 2011, pp. 247–262.

[12] T. Xu and Y. Cai, "Feeling-based location privacy protection for location-based services," in *Proceedings of the 16th ACM conference on Computer and communications security*, ser. CCS '09, 2009, pp. 348–357.

[13] H. Takabi, J. Joshi, and H. Karimi, "A collaborative k-anonymity approach for location privacy in location-based services," in *5th Int. Conference on Collaborative Computing: Networking, Applications and Worksharing*, 2009, pp. 1–9.

[14] M. Benisch, P. G. Kelley, N. Sadeh, and L. F. Cranor, "Capturing location-privacy preferences: quantifying accuracy and user-burden tradeoffs," *Personal and Ubiquitous Computing*, vol. 15, no. 7, pp. 679–694, Oct. 2011.

[15] A. B. Brush, J. Krumm, and J. Scott, "Exploring end user preferences for location obfuscation, location-based services, and the value of location," in *Proceedings of the 12th ACM international conference on Ubiquitous computing*, ser. Ubicomp '10, 2010, pp. 95–104.

[16] Team LBE (http://www.lbesec.com/). LBE Grivacy Guard. [Online]. Available: https://play.google.com/store/apps/details?id=com.lbe.security.lite

[17] S. Thiele (CollegeDev). PDroid 2.0. [Online]. Available: https://github.com/CollegeDev/PDroid2.0_Manager

[18] W. Enck, P. Gilbert, B.-G. Chun, L. P. Cox, J. Jung, P. McDaniel, and A. N. Sheth, "Taintdroid: an information-flow tracking system for realtime privacy monitoring on smartphones," in *Proceedings of the 9th USENIX conference on Operating systems design and implementation*, ser. OSDI'10, 2010, pp. 1–6.

[19] P. Hornyack, S. Han, J. Jung, S. Schechter, and D. 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*, ser. CCS '11, 2011, pp. 639–652.

[20] R. Beede, D. Warbritton, and R. Han, "Myshield: Protecting mobile device data via security circles," University of Colorado Boulder, Tech. Rep. CU-CS-1091-12, January 2012.