Data Minimization in a mobile crowd sensing scenario

Vladimir Alekseychuk
Technische Universität Berlin
Berlin, Germany

ABSTRACT

During the development of the internet, privacy awareness was not able to grow at the same speed as the technical possibilities to infringe on it. A big milestone were the leaks presented by Edward Snowden that revealed the kept-in-secret scope of one nation spying on their citizens and those of other nations around the world. They have lead to a noticeable rise of public awareness and the creation of the General Data Protection Regulation (GDPR) in the European Union. However, not every developer who wants to bring his app to market has studied privacy enhancing measures intensively, even if they actually want to protect their users privacy. Therefore a usable and easily implementable system is needed to help developers on a large scale that want to protect the privacy of their users without doing a deep dive into the field of privacy enhancing research. Exactly such framework is presented in this work and evaluated using a made up example location based crowd sensing service. The framework features several obfuscation functions for a variety of sensor and location data. There functions can easily be configured following a extensive documentation on GitHub [5]. For further improvement or own modified versions of it, its source code is published for all who are interested.

KEYWORDS

Privacy by Design, Data Minimization, Crowd Sensing

ACM Reference format:
Vladimir Alekseychuk. 2019. Data Minimization in a mobile crowd sensing scenario. In *Proceedings of Privacy by Design Conference, Berlin, Germany, July 2019 (Berlin'19),* 7 pages.
https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUCTION

Nowadays most people are using mobile devices everyday and they have become our constant companion. While these little helpers improved over time, so did their ability to collect all kinds of sensor data. It is mostly used to provide various services, from which a big percentage works with the users location and are thus called location based services (LBS). Though improving everyday life, these services also have lead to a substantial loss in privacy for the user, since the service providers have become all-knowing when it comes about the data of their clients. In times of Big Data, even seemingly uninteresting data can be used to infer very private information. This knowledge is often used to deliver personalized advertisements and can even enable a third party to strategically manipulate our goals through e.g. social networks or a social credit system. There has been a lot of research into many diverse ways to mitigate or stop the loss of privacy. However, most of this research has lead to no available application improving software. Therefore this work proposes and implements a reusable software building block for developers to enhance the way they collect sensor data for their crowd sensing services in a way that preserves the users privacy in the best possible form. The focus of this system is to be easily understandable and implementable so that it will actually be used. The challenge of a working solution for sensor data anonymization lies within the use case itself. While most data anonymization is focused on large data sets and their anonymization for publication and further processing, it ignores the fact that the data has already been collected at some point and a controller (as described by the GDPR [1]) has already gathered personal information about their data subject. Therefore the aim of this project could be a first step to create a trusted and regularly used open-source data minimization packet that enables crowd sensing services to keep their users privacy protected without having to spend many resources into development of own implementations. Since this trusted component needs to run inside an untrusted data collection application, it is impossible to actually make sure that it is not modified and properly used. Still, for service providers that do not have malicious intents, the results of this work could be very helpful.

The first part of this work will examine the already present theoretical work on data minimization in section 2 to determine what has already been done and what the state of the art in privacy preserving technologies is. After that, a concept for a data minimization building block will be sketched and goals for a generic and reusable implementation will be defined in section 3. Based on these goals, the proposed system with all its functionalities is presented in section 4. To demonstrate its effectiveness in a realistic environment, a demonstration application was developed that represents a typical crowd sensing service that uses the proposed data minimization blocks in section 5. In the following section, the results of working with the data minimization blocks are evaluated 7. Further possible improvements of the implementation and concept are presented in section 8.

2 RELATED WORK

The idea of data minimization comes from the concept of Privacy by Design by Cavoukian [7]. It is an important cornerstone of the principle of "privacy as the Default". Cavoukian states that "collection of personally identifiable information should be kept to a strict minimum." Also, the collection of *any* data should be limited to what is necessary for specified purposes [7]. These demands are seen as important guidelines throughout this work. Schaar refined this concept a bit further by also specifying in concept of transparency that data subjects have to be informed about how the system works [13].

Ardagna at. al. presented a collection of means to obfuscate locations by either increasing the radius of inaccuricy or shifting

Berlin'19, July 2019, Berlin, Germany
2019. ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

the center of a measured area [6]. Both approaches can also be combined to achieve even better obfuscation.

A good and very early attempt at building an architecture that supports these guidelines, using multiple methods or algorithms in the scope of Location-Based-Services (LBS) was presented by Gruteser et. al. [9]. Already in the year 2003 before the spread of smartphones and GPS enabled devices they devised an architecture that is based on grouping of database entries that provides generalization to achieve k-anonymity as proposed by [12]. The grouping was achieved by applying a spatial as well as a temporal decrease in accuracy. It is a very similar method of generalization as used in the k-anonymity approach. The main difference is that instead of anonymizing an existing, non-anonymous database, the data is obfuscated already when it is collected. What they could not foretell was the development of GPS sensors to become highly accurate and the correlated rise in popularity of services that require an accurate position to work. Since their approach was based on a decrease of spatial accuracy, it is not very usable for many of todays use cases.

A more modern approach that is very similar to the here proposed system has been done by AppPET [14]. It is a framework that supports developers at creating privacy preserving applications. While the idea and the overall structure of both systems is very similar, there is a key difference: availability. The promise of the AppPET project was that at the end of their funding and project lifecycle, all their code would be made public to be used by developers as intended. As of June 2019, no source code can be found while the funding has run out in January 2019. Therefore the proposed system in this work will be developed on a public service (github.com) so that developers can use it right away and the promise of an available and easily usable framework will be kept.

There is also a number of commercial privacy enhancing kits sch as the "Privacy Kit" by the "Privacy Company" [3]. Unfortunately, this kit costs money that application developers might want to invest in other processes. For smaller companies, the cost to buy a privacy kit are higher than the expected payoff. This is why a free framework that is open-source and can be reviewed and improved by anyone is the best way to promote data privacy in a widespread manner.

3 GOALS

This work focuses on the privacy principle of data minimization by Cavoukian, presented in the previous section. Schaar states that "Data processing systems should be designed and selected in accordance with the aim of collecting, processing and using to personal data at all or as little personal data as possible" [13]. This basically means that all data should be stripped of all unnecessary personal identifiers and pseudonymization and anonymization should be applied where possible. Another demand of this principle is that data should be minimized as early as possible [13]. Ensuring real anonymity in a crowd sensing scenario is difficult, since users might want to be rewarded for their services and for that each data packet needs to feature an identifier. This is why the goal of this project is to obfuscate all measured data to a point where even when being able to infer a natural person from a pseudonymous identifier, the collected data does not allow for inferring personal information from it. Additionally, the support of k-anonymity should be implemented in a server-based component. Since there are many approaches to obfuscate location data and obfuscation of simple numeric values is easy enough, all there needs to be is a framework that can bring all those methods together in an opely accessible manner. That alone is not enough, though. Such a framework needs to be generic enough to fit all possible use cases, be easily accessible and understandable to all interested programmers so that there is absolutely no reason not to use it. In this work, a typical location based crowd sensing scenario, where multiple (usually) mobile clients collect data and send it to a server that uses it for further processing, is regarded. A possible result of such crowd sensing could be a map of noise levels around a city to find out, which neighborhood is the most interesting in terms of quietness. A more traditional "crowd" sensing scenario is the weather prediction, where weather stations across the world measure data such as temperature, humidity and atmospheric pressure to feed a prediction model. To have some kind of anchor point when thinking about the countless possible scenarios of data collection, some assumptions were made.

(1) First of all, since sensor data is collected in a location based scenario, all data is linked to coordinates. There is no use to send data without a spatial context, since it would not follow a location based approach.
(2) Second, since sensor readings are dependent on the spatial context, this context (e.g. coordinate) needs to be collected in short enough intervals to stay relevant. Collecting data for several minutes before getting a GPS fix on the current location makes the data unusable, if the user has moved a considerable distance since starting the collection. To simplify things, it is just assumed that all data was collected within a small range of the coordinate. When collecting coordinates in short intervals, this problem does not occur.
(3) In an effort to try to make sure that obfuscated data is not tempered with after leaving the minimization block, it is internally buffered and sent to a specified server, controlled by the service provider. With this, a problem comes up: What data format should be used so that most service providers would accept to use it? In this work, it is simply assumed, that JSON is good enough for most information types.
(4) The minimization block internally uses mostly values of the type *Double*. This is also done when no floating point numbers are processed. This is because it is not clear what kind of data may be passed from external modules. Since double is able to represent both integer and floating point numbers, it is assumed that this representation is seen as fitting enough.

4 DATA MINIMIZATION BLOCKS

The data Minimization Blocks consist of two parts. The first one is supposed to run on mobile clients as part of the crowd sensing application. A typical data flow without this block would be that the service application collects all necessary (and maybe more) data from the client's phone sensors to optionally do some calculations based on them and then package all information for sending it to a server. There, all data is merged and computed to create a service that represents measured and inferred information on a

geographical map. Now with using the client-side minimization block, the collected data is passed to the minimization module that can be configured using a JSON coded *String* that includes all necessary settings. It will be presented in section 4.3. All data passed to the minimization block is obfuscated and then sent to the server. This ensures in theory, that no additional data like extra identifiers can be added to the sent package. In practice though, it is not possible to block the application logic from opening a side-channel and sending uncontrolled data. For data sent through the minimization block, there is also a feedback function that returns information about quantity and quality of all data in a human readable format. With this, it is possible to inform the user about the impact of using data minimization and give them back full control about their collected data. Since it is possible to change the minimization settings on the fly, a user might want to adjust their settings according to personal preferences dependent on e.g. time of day or location.
The other component is a minimization block inside the server logic that is responsible for two tasks: One is to receive data packets and refuse to forward metadata linked to the transmissions. It also applies a form of k-anonymity [12] which is further explained in section 4.1. The other task is the creation of a simple peer-to-peer network where clients know each others IP addresses and use them to apply identity obfuscation, which is explained in the next section. This solution has the downside that clients need to register themselves regularly, since their IP addresses might change. When doing this, even when communicating solely with the trusted P2P broker of the minimization block, the server environment (e.g. routers or firewalls) is able to create a list of its user's addresses. This contradicts the idea of masking addresses in a direct manner. Other solutions for building such a network would be using Bluetooth [10], public WiFi networks or ad-hoc Wifi networks. All of these solutions have even greater downsides (e.g. limited range) or rely on non-fulfillable conditions, mainly an extremely large user base.

4.1 Data Minimization

Depending on what settings are specified, there are different obfuscation functions that get applied to data before it is sent to the server. Since it might come in different formats and with different meanings, only the most common sensors found in modern smartphones are supported. They are: location sensor, gyroscope, acceleration sensor, brightness sensor and compass. These sensors give data in three data types:

- Location sensor: A tuple containing the latitude and longitude. Accuracy is not regarded due to reasons of simplification.
- Gyroscope and acceleration sensor: Three values (*Triple*) for the x-, y- and z-Axis in a 3D environment.
- Brightness and compass sensor: A single floating-point number each.

Adding support for other sensor readings should not pose a difficult task, since the code for the supported sensors can be used as a template for any other reading type and the GitHub documentation [5].
Based on set preferences, the following obfuscation functions are applied:

- Temporal obfuscation, as presented by Gruteser et al. [9]. All data is held back with a random delay so it is impossible to tell *when* a user has collected the data and e.g. driven a certain route from home to workplace. It may happen could that packets may change their order as a result of randomized delays. In some cases, where new data is linked to the previously received data packet, a random order might break the functionality of the service. Another possible configuration is to send data only at fixed times during the day, e.g. always at 12 AM and 12 PM, maintaining their order. Both settings are equal in terms of temporal obfuscation, as long as a high enough maximum delay and few enough sending times with high distances between each other are specified.
- Spatial obfuscation by increasing the radius [6], which leads to GPS coordinates being altered to lie somewhere within a given radius around the real coordinates. For services that only need an approximate position (e.g. street illumination in the example use case), the exact position of the user is masked with a position e.g. within a 20 metres radius.
- GeoFences can be specified to block all data collection and thus further protect the location of users. The used simplified implementation allows to specify a list of coordinates with radii that each form a geographical circle. If data is collected from within one of these circles, it is simply dropped and forgotten by the minimization block. Such a feature might be highly welcome for users who do not want to have data saved that states their presence at an area and could be used to infer their personal habits. Such inferred information can have an impact on people's lifes and is especially critical, if it is inaccurate. Imagine a person working at a grocery store that is located in the red light district. With incomplete data, an analyzing party could infer that this person is a prostitute (or similar). Simply masking the district one lives in is of course also a valid reason to apply this function.
- Rounding of all numerical values. It is basically spatial obfuscation in a non-spatial context. A number can be specified to which multiples of a number (e.g. [..., -5, 0, 5, 10, ...] for '5') incoming numerical values are rounded. This feature is useful for sensors whose readings do not have to be extremely accurate, e.g. road illumination where it is enough to know whether the illumination is *bright*, *acceptable*, *dark*, or *non-existent*.
- Sending only maximum and minimum value: When collecting sensor readings in intervals of milliseconds, each data packet might end up containing thousands of data samples where only the highest and smallest number might be actually necessary. So instead of sending thousands of numbers, only two are sent.
- A further minimization of data can be achieved by sending the calculated average of all numbers. From all proposed numerical obfuscation functions, this method applies the best data minimization, since it maps x different values to just one.
- Identity obfuscation follows a decentralized approach of hiding the IP address by using a peer-to-peer network in which users exchange their data packets that would be sent to the server. When receiving a data packet from another user,

the receiver forwards the data to the service server using their own IP address. This could be taken as far as that only one user is responsible for uploading data and masking all other addresses. Of course, in that case this user needs to be trusted as well. This problem is not further addressed in this work, since this module could not be completed because of internal issues within the development team. After applying such obfuscation, it would be then impossible for the server to say which user owns which and how many IP addresses and sends how many data packets, in case no identifiers are used.

A server-side component would also be able to implement k-anonymity [12]. For that, when accepting new data packets, each coordinate is reviewed regarding to the existing database. If it can be inserted without breaking the k-anonymity condition, the data is simply added to the database. In case there are no identifiers linked to the data, it can be seen as anonymous and the condition is always given. If there are identifiers, they can be considered private information that needs to be protected. When adding the information would break the k-anonymity condition, it could either be further obfuscated or held back inside a buffer until enough other information is collected that together forms a block with k identical non-private fields.

4.2 Implementation

For implementing the client-side minimization block, the programming language of choice was Java, since it features the Java Native Interface (JNI) which enables communication to and from C, C++ and assembly. The whole block is a Class that features local buffers for incoming data, public functions to receive new data and private functions for obfuscation and packaging data for the purpose of sending it to a specified server. Before working with data, the following function should be executed:

```
public void setSettings(String JSON)
```

The passed String needs to be a valid coded JSON object that is further described in the following section. If this step is skipped, default settings will be applied leading to the highest implemented obfuscation level. It includes *spatial obfuscation*, *rounding*, *averaging*, *temporal obfuscation* and *identity obfuscation*. When adding new data, the first processing step is decreasing its accuracy, if the settings specify it. If this step is enabled, numerical values get rounded according to the description in section 4.1. Only after this, they are added to the corresponding local buffer. These buffers fill up, until a coordinate is passed to the class. Such approach follows directly the idea of data minimization at the earliest possible stage schaar. Internally, some helpful classes were defined from which *Coordinate* is one of them. Its members are *latitude, longitude*, a *radius* for usage in GeoFences and *List* types for sensor data. Before adding a coordinate to the local buffer, the sensor data buffers are either averaged, analyzed for maximum and minimum values or directly flushed into the lists, depending on the set preferences. As a result, each *Coordinate* type holds all sensor readings (or the results of their analysis) since the last coordinate was added. With this representation, it is easy to link sensor readings to the spatial area they were collected and to divide the coordinate buffer into packets for transmission. In case the collected coordinate conflicts with a GeoFence, meaning it is inside the specified radius, the coordinate is not added to the coordinate buffer and the sensor data buffers are also cleared, since the data may be collected inside the forbidden zone. As long es there is an instance of the Minimization Block, a *Timer* is responsible of sending data packets, in case random delays were added or the data is supposed to be sent at fixed times. If no temporal obfuscation is applied, coordinates are sent to the server, as soon as they are collected and filled with sensor data.

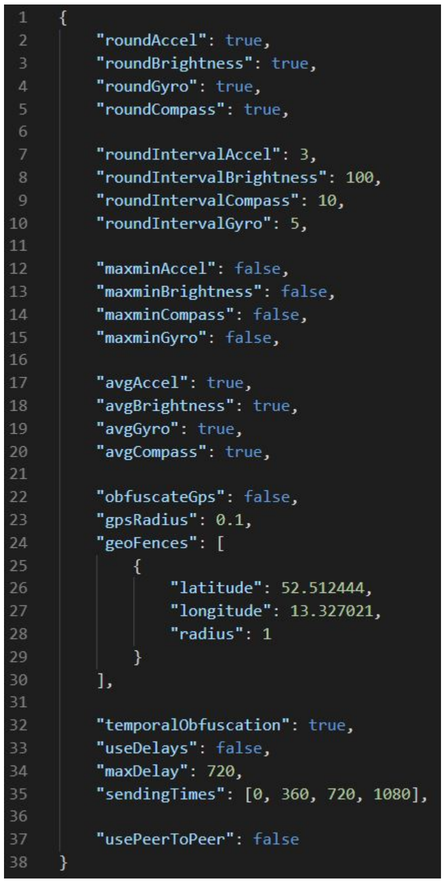
Unfortunately due to issues within the development team, work on the server-side implementation has never started. The minimization component is able to accumulate data, obfuscate it and generate a feedback string when it would have be sent.

4.3 Settings

As mentioned in the previous sections, the minimization block needs to be configured to provide the preferred level of data obfuscation. This is done by passing it a JSON coded String that contains values for enabling and disabling certain functions and the necessary parameters for those. For rounding functions, a rounding interval as specified in section 4.1 is needed. GeoFences are also part of the settings. The value for the key *geoFences* is a JSON array populated with JSON objects, representing coordinates and their radius in kilometres. An example settings file can be seen in Figure 1. A more detailed explaination of the settings parameters can be found on GitHub **??**.

5 EXAMPLE USE CASE

For this project, one use case needed to be created to demonstrate that the proposed framework actually works. For that, the idea of maps for self-driving cars was an interesting scenario. In the work of Geiger et. al.[8], a format for such maps was proposed, in which the center of a driving lane is a path that is being followed by autonomous cars. Since one single measurement car might not drive exactly in the center of a driving lane or GPS reception might be inaccurate, it would be useful to accumulate the data collected by dozens, hundreds or even thousands of users who drive this route regularly. For this, users would only need to center their mobile phone inside the car and start the crowd sensing service. The collected data would form one average driving line that is either centered on the lane or takes the optimal route when e.g. the outer side of the road is in a bad condition. Also, the more accurate the location data, the more valuable such information is. It is in fact worthless, if it is known that the GPS sensor had an accuracy worse than 1.5 metres, since standard driving lanes have a width of at least 3 metres [2]. Then, it would be impossible to say which driving lane was actually measured. Unfortunately, such demands for accurate data pose a significant threat for individual privacy. From just a few location points and their timestamps of arrival, one could infer the speed a person was travelling with and maybe calculate whether the person is rather risk-friendly or careful. Based on such data, in case it would be sold or stolen, the users of such applications would become analyzed by people or companies that they would never want to have their private data. But location data is not the only interesting aspect of such maps. There are countless attributes that can be linked to each road segment or position and that could



```json
{
    "roundAccel": true,
    "roundBrightness": true,
    "roundGyro": true,
    "roundCompass": true,

    "roundIntervalAccel": 3,
    "roundIntervalBrightness": 100,
    "roundIntervalCompass": 10,
    "roundIntervalGyro": 5,

    "maxminAccel": false,
    "maxminBrightness": false,
    "maxminCompass": false,
    "maxminGyro": false,

    "avgAccel": true,
    "avgBrightness": true,
    "avgGyro": true,
    "avgCompass": true,

    "obfuscateGps": false,
    "gpsRadius": 0.1,
    "geoFences": [
        {
            "latitude": 52.512444,
            "longitude": 13.327021,
            "radius": 1
        }
    ],

    "temporalObfuscation": true,
    "useDelays": false,
    "maxDelay": 720,
    "sendingTimes": [0, 360, 720, 1080],

    "usePeerToPeer": false
}
```

Figure 1: Exmaple settings file specifying all possible parameters to configure the minimization block

have a major influence on route planning. Two of such aspects are also measured by the example application. One is the brightness of the road illumination. Of course, such measurement is mostly interesting at night or in tunnels and since it drastically changes during the day, a timestamp could also needed to be linked. For this, the accuracy of the data is not very essential. It would not be a loss for the original cause of data collection, if the brightness would not be extremely accurate in terms of data samples and measured numbers. The timestamp also does not need to to be as accurate as to the millisecond of the day. It would be enough to know which month and which daytime in hours the data was collected in. For reasons of simplicity, the timestamps of brightness values are only mentioned but not implemented. The other collected data is actually a collection of two sensors: accelerator data to measure at which locations vehicles frequently needed to brake (e.g. pedestrian crossings), when they switched lines or got stuck in a traffic jam. The gyroscope sensor is needed to find out, in which position the phone is resting to know whether an acceleration of -2 means a decrease of speed or the contrary, because the phone is turned by 180 degrees.

Since this scenario is was only created to prove the functionality of the data minimization framework and to collect necessary data, it is allowed to have possible logic holes.

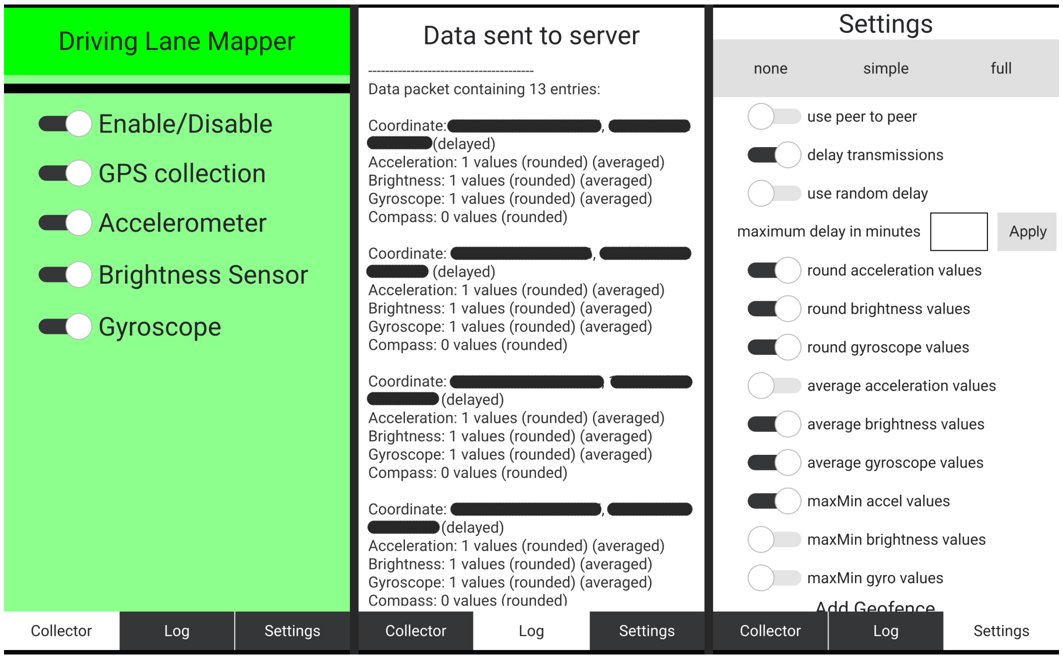
Even though the above mentioned data does not always need to be as accurate as possible, it is usually collected in such a manner. Two reasons come to mind why that could be. One is the cost-to-gain factor of implementing data minimization functions and possible lack of knowledge in that field of the responsible developer or development team. When the customers might not really care about their data, which is often the case[11], and acquiring privacy expertise costs money, it could be the logical conclusion to skip this step (if profits are the only thing that matter). Another, even less flattering explanation might be the explicit intent to collect as much data as possible in terms of what is allowed legally. Such data could then be used not only to provide the service functionality but also to “improve one’s service quality” and possibly sell it to analysis companies. There is no other way to stop such a service provider than by stopping to use its services. But for those providers who actually want to keep their customers privacy intact and simply are not able to do so, this work introduces the Data Minimization Blocks.

6 DATA MINER EXAMPLE

To prove that the client-side minimization block is working as intended, the above mentioned scenario was implemented as an Android crowd sensing application. It was programmed using QT QML and C++ to demonstrate that the module created with Java is also compatible with C++. In the implementation, QML was used to display a user interface for enabling specific sensors and specifying the obfuscation preferences. The three pages of the application can be seen in figure 2. The first page is used to enable and disable single sensors. The center page is used to display a feedback text about what data was sent to the server and which obfuscation functions were applied. The right page is reserved for specifying settings. Here, a service provider could decide which settings are not displayed to the user due to necessities for the service to function. In this case, no switch for spatial obfuscation was added, since it would have made all collected data unusable. The C++ part of the application was used to access the settings stored in a file on the phone, because QML is unable to do that. Since this application is for demo purposes only, security features like encryption of the settings file were not implemented. The same applies to optics and usability. Also C++ was of course used to communicate with the Minimization Block via JNI.

7 EVALUATION

During testing, the problem of a "Privacy Bottleneck" came up. When collecting accurate location data and mapping other sensor readings that do not depend on an accurate location, they are still mapped to the accurate coordinates. In case of the brightness reading, there would even be information about the hour of day (if it were implemented) in which the coordinate was collected. This clearly disagrees with the previously stated goal to reduce data accuracy only to what is necessary. It could be solved by slightly changing the internal representation. Also this presented module might not work for all possible crowd sensing scenarios, but



Driving Lane Mapper

[✓] Enable/Disable
[✓] GPS collection
[✓] Accelerometer
[✓] Brightness Sensor
[✓] Gyroscope

```
Data sent to server

---------------------------------------
Data packet containing 13 entries:

Coordinate:[redacted], [redacted]
[redacted](delayed)
Acceleration: 1 values (rounded) (averaged)
Brightness: 1 values (rounded) (averaged)
Gyroscope: 1 values (rounded) (averaged)
Compass: 0 values (rounded)

Coordinate: [redacted], [redacted]
[redacted] (delayed)
Acceleration: 1 values (rounded) (averaged)
Brightness: 1 values (rounded) (averaged)
Gyroscope: 1 values (rounded) (averaged)
Compass: 0 values (rounded)

Coordinate: [redacted], [redacted]
[redacted](delayed)
Acceleration: 1 values (rounded) (averaged)
Brightness: 1 values (rounded) (averaged)
Gyroscope: 1 values (rounded) (averaged)
Compass: 0 values (rounded)

Coordinate: [redacted], [redacted]
[redacted](delayed)
Acceleration: 1 values (rounded) (averaged)
Brightness: 1 values (rounded) (averaged)
Gyroscope: 1 values (rounded) (averaged)
Compass: 0 values (rounded)
```

Settings

none simple full

[ ] use peer to peer
[✓] delay transmissions
[ ] use random delay
maximum delay in minutes Apply
[✓] round acceleration values
[✓] round brightness values
[✓] round gyroscope values
[ ] average acceleration values
[✓] average brightness values
[✓] average gyroscope values
[✓] maxMin accel values
[ ] maxMin brightness values
[ ] maxMin gyro values
Add Geofence

Figure 2: Interface of the example use case implementation. The first page is for enabling data collection, the second page provides a feedback of sent data and the last page offers means to customize data obfuscation as much as the service provider allows. Masking of latitude and longitude values was done by hand for privacy reasons

for the most typical ones, it should be enough. By reviewing the open-source code, it is possible to add support for other sensors or the results retrieved from functions before using the minimization block. As mentioned before, the project is not quite finished. Another developer had the task to develop the server-side module and networking capabilities but work never started. This is why the final result is incomplete. The module in its current state is only able to collect data, obfuscate it and return the feedback text when the data would be sent.

On the more positive side, it can be said that what is done is also working and publicly accessible on GitHub [5]. It includes source code, the final *.jar* file, the source code of the example application, an example settings file and a documentation on all functions and settings parameters. With this, a service developer could use the code or parts of it to improve their own service. The example implementation also proves that the Java implementation is compatible with C++. the necessary parameters on how to establish the JNI link are also explained in the GitHub documentation. One feature that the author could not observe in other projects was the feedback function so that users actually know what data they are sending to a service provider. It gives control over the data back to the users who have collected it and fulfils the transparency concept by Schaar schaar.

8 FUTURE WORK

Of course in future revisions of the project, the networking capabilities should be added, as well as the server-side component that includes the k-anonymity functionality. Also, data support is currently purely numerical. A support for the removal of meta-data in images would be a useful extension. Regarding the feedback function, the returned text could be improved or split into two versions, from which one offers more detailed data than just the number of data samples and the applied obfuscation. To foster the understanding about their data, service users could be displayed a small text explaining what kind of private information can be inferred from their seemingly unimportant data. This could lead to a whole educational/informative branch of functions inside the client-side minimizing block. Such a branch could be combined together with approaches of privacy nudges, similar to Privacy Nudges, presented by Acquisti [4].

9 CONCLUSION

As a conclusion in can be said that some work has been done, but more work needs to be invested so that this project results in a truly reusable component.
The GitHub repository can be accessed at: https://github.com/etho2183/peng_data_minimization

REFERENCES

[1] General data protection regulation (gdpr), art. 4, paragraph 7.
[2] National association of city transportation officials, urban street design guide.
[3] Privacy company, privacy kit; a helpful tool, 2019.
[4] Acquisti, A., Adjerid, I., Balebako, R., Brandimarte, L., Cranor, L. F., Komanduri, S., Leon, P. G., Sadeh, N., Schaub, F., Sleeper, M., et al. Nudges for privacy and security: Understanding and assisting users' choices online. *ACM Computing Surveys (CSUR) 50*, 3 (2017).
[5] Alekseychuk, V. Reusable data minimization module for client side data collection, 2019.
[6] Ardagna, C. A., Cremonini, M., Damiani, E., Di Vimercati, S. D. C., and Samarati, P. Location privacy protection through obfuscation-based techniques.

In *IFIP Annual Conference on Data and Applications Security and Privacy* (2007), Springer, Berlin, Heidelberg, pp. 47 – 60.

[7] Cavoukian, A. Privacy by design: The 7 foundational principles. *Information and Privacy Commissioner of Ontario, Canada 5* (2009).

[8] Geiger, A., Lauer, M., Mooksmann, F., Ranft, B., Rapp, H., Stiller, C., and Ziegler, J. Team annieway's entry to the 2011 grand cooperative driving challenge. *IEEE Transactions on Intelligent Transportation Systems 13* (2012), 1008 – 1017.

[9] Gruteser, M., and Grunwald, D. Anonymous usage of location-based services through spatial and temporal cloaking. In *1st international conference on Mobile systems, applications and services* (2003), pp. 31 – 42.

[10] Johansson, P., Kazantzidis, M., Kapoor, R., and Gerla, M. Bluetooth: an enabler for personal area networking. *IEEE Network 15*, 5 (2001), 28 – 37.

[11] Li, N., and Chen, G. Sharing location in online social networks. *IEEE Network 24* (2010), 20 – 25.

[12] Samarati, P., and Sweeney, L. Protecting privacy when disclosing information: k-anonymity and its enforcement through generalization and suppression. Tech. rep., SRI International, 1998.

[13] Schaar, P. Privacy by design. *Identity in the Information Society 3* (2010), 267 – 274.

[14] Sy, E., Mueller, T., Marx, M., and Herrmann, D. Apppets: a framework for privacy-preserving apps. In *Proceedings of the 33rd Annual ACM Symposium on Applied Computing* (2018), ACM, pp. 1179 – 1182.