Floating Car Data from Smartphones: What Google and Waze Know About You and How Hackers Can Control Traffic

Tobias Jeske
Institute for Security in Distributed Applications
Hamburg University of Technology
21079 Hamburg, Germany
Email: research@tobiweb.com

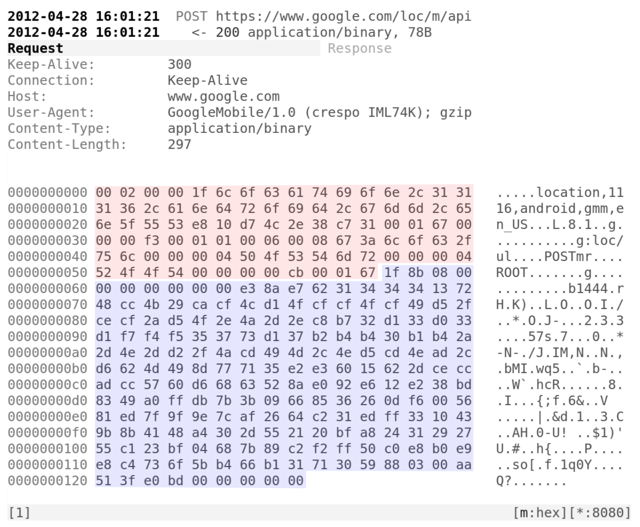
***Abstract*—In recent years, a trend of using real-time traffic data for navigation has developed. Google Navigation and Waze, for instance, generate traffic data from movement profiles of smartphones. In this paper we tackle the question to which extent it is possible for Google and Waze to track the smartphone and its owner. Furthermore, we show how wireless access points and smartphones acting like wireless access points can be located around the world. In addition to the privacy issue, we examine whether the authenticity of traffic data can be guaranteed. We demonstrate in practice how hackers can take control of navigation systems and, in the case of a wide distribution of floating car data, can actively control the traffic flow. At the end we present a practical protocol preventing such attacks and at the same time preserving the user's privacy. The protocol has been implemented on different hardware platforms and benchmark results are given.**

I. Introduction

Today, navigation devices frequently receive traffic reports on the Traffic Message Channel (TMC) [23]. TMC messages have several sources: the police, permanently installed sensors like traffic cameras or inductive loops, and traffic reports of volunteers. Radio stations transmit traffic information in the non-audible range of the FM frequency band. TMC is widespread. However, it has two major drawbacks. On the one hand, traffic reports are often out of date if they are processed by the navigation device. The traffic reports are, for the most part, still edited by humans and are not automatically generated and sent. On the other hand, the transfer rate of 60bit/s is low, consequently only approximately 10 messages can be transmitted per minute.
In 2007, the two security researchers Andrea Barisani and Daniele Bianco conducted a study on the security of TMC, showing how counterfeited TMC messages can be sent to navigation devices [3]. The attack is possible because TMC data is not transmitted in an encrypted way. However, it is necessary that the navigation devices are in the range of the transmitter.

In 2007, Google extended Google Maps by adding Google Live Traffic, the visualization of traffic information in real time [8]. In contrast to TMC, Google uses position data of smartphones with the Android operating system [2], which results in a significantly faster mapping of the traffic flow. This data is called floating car data (FCD). Position data is determined by the navigation system or, as in the case of Google Live Traffic, by the smartphone and is transmitted to the service provider via a mobile phone connection. Compared to TMC, this allows the generation of traffic information in real time.
After activation by the user, the traffic flow on Google Maps is visualized by the three colors red, orange and green. A red road points to a traffic jam or stop-and-go traffic, orange indicates heavy traffic and green points to clear roads. At first Google was only able to assess the traffic flow on some main roads in a few cities in the United States. Later other cities followed and today traffic information is available in many countries.
In 2007, as part of the Google Street View project, Google took pictures from the street perspective with the help of specially equipped vehicles. At the same time, the vehicles saved position information of wireless routers. Routers can be clearly identified by their MAC address and SSID [11].
The traffic flow data is not only visualized by Google Maps but, since 2011, has also been used to optimize route calculation in Google Navigation [18]. Traffic jams can therefore be detected in real time and avoided.

Waze is a free GPS application, which also uses FCD of smartphones in order to generate traffic information [24]. The application can be installed free of charge on all major smartphone operating systems like Android, IOS, Windows Mobile, Symbian and BlackBerry. The number of users is continuously increasing and, end of 2012, amounted to approximately 36 million users [24]. The application is in the top 20 of the most often downloaded free apps in the iTunes Store. In addition to navigation, users can add new roads and can, for example, report accidents, traffic jams and speed traps directly via the Waze-App. Every app sends the user's position to Waze so that Waze can generate traffic information in real time. In this case, the position data of smartphones is also used to optimize the navigation.


```
2012-04-28 16:01:21  POST https://www.google.com/loc/m/api
2012-04-28 16:01:21    <- 200 application/binary, 78B
Request                                            Response
Keep-Alive:         300
Connection:         Keep-Alive
Host:               www.google.com
User-Agent:         GoogleMobile/1.0 (crespo IML74K); gzip
Content-Type:       application/binary
Content-Length:     297


0000000000 00 02 00 00 1f 6c 6f 63 61 74 69 6f 6e 2c 31 31   .....location,11
0000000010 31 36 2c 61 6e 64 72 6f 69 64 2c 67 6d 6d 2c 65   16,android,gmm,e
0000000020 6e 5f 55 53 e8 10 d7 4c 2e 38 c7 31 00 01 67 00   n_US...L.8.1..g.
0000000030 00 00 f3 00 01 01 00 06 00 08 67 3a 6c 6f 63 2f   ..........g:loc/
0000000040 75 6c 00 00 00 04 50 4f 53 54 6d 72 00 00 00 04   ul....POSTmr....
0000000050 52 4f 4f 54 00 00 00 00 cb 00 01 67 1f 8b 08 00   ROOT.......g....
0000000060 00 00 00 00 00 00 e3 8a e7 62 31 34 34 34 13 72   .........b1444.r
0000000070 48 cc 4b 29 ca cf 4c d1 4f cf cf 4f cf 49 d5 2f   H.K)..L.O..O.I./
0000000080 ce cf 2a d5 4f 2e 4a 2d 2e c8 b7 32 d1 33 d0 33   ..*.O.J-...2.3.3
0000000090 d1 f7 f4 f5 35 37 73 d1 37 b2 b4 b4 30 b1 b4 2a   ....57s.7...0..*
00000000a0 2d 4e 2d d2 2f 4a cd 49 4d 2c 4e d5 cd 4e ad 2c   -N-./J.IM,N..N.,
00000000b0 d6 62 4d 49 8d 77 71 35 e2 e3 60 15 62 2d ce cc   .bMI.wq5..`.b-..
00000000c0 ad cc 57 60 d6 68 63 52 8a e0 92 e6 12 e2 38 bd   ..W`.hcR......8.
00000000d0 83 49 a0 ff db 7b 3b 09 66 85 36 26 0d f6 00 56   .I...{;f.6&..V
00000000e0 81 ed 7f 9f 9e 7c af 26 64 c2 31 ed ff 33 10 43   .....|.&d.1..3.C
00000000f0 9b 8b 41 48 a4 30 2d 55 21 20 bf a8 24 31 29 27   ..AH.0-U! ..$1)'
0000000100 55 c1 23 bf 04 68 7b 89 c2 f2 ff 50 c0 e8 b0 e9   U.#..h{....P....
0000000110 e8 c4 73 6f 5b b4 66 b1 31 71 30 59 88 03 00 aa   ..so[.f.1q0Y....
0000000120 51 3f e0 bd 00 00 00 00 00                        Q?.......

[1]                                                          [m:hex][*:8080]
```

Fig. 1. Screenshot mitmproxy
The MASF header is marked red and the compressed protobuf payload is marked blue

Due to the high growth rates in the smartphone market, Google and Waze can expand their system and can achieve a higher accuracy through more smartphones and, therefore, more sensors in the field. It is expected that, in the future, traffic data will be available on almost all roads.

A. Requirements

When considering the traffic analysis on the basis of location information of smartphones, there are two main requirements:

- **Privacy:** The smartphone owners are interested in a high degree of privacy. User tracking by service providers such as Google or Waze is generally considered as undesirable by the user.
- **Authenticity:** The service provider is interested in the correctness of the data. "Malicious smartphones" sending wrong location data should be excluded from the calculation of the traffic flow. Incorrect traffic data influences the route planning and thus the user's navigation. If the data is used for navigation by a huge number of users, hackers can significantly affect the traffic flow.

In the next section, we explain the Google protocol and in section III the Waze protocol. We evaluate the two protocols in section IV and show to which degree the requirements "privacy" and "authenticity" are met. In section V we present a protocol which fulfills the privacy and authenticity requirement. The protocol has been implemented on different hardware platforms and benchmark results are given before we draw conclusions in section VI.

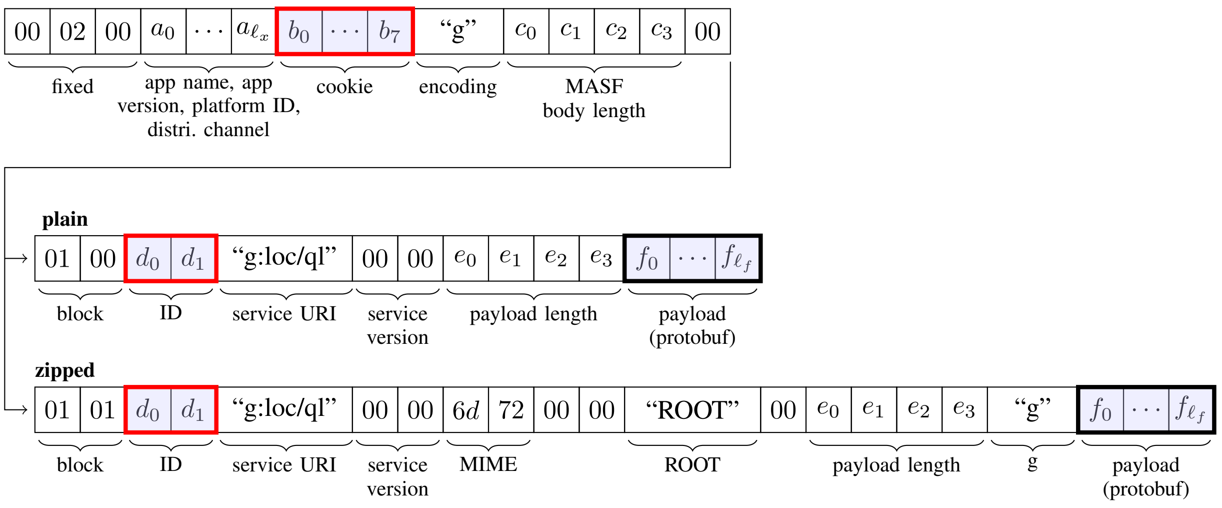
II. Google Protocol

The analysis of the protocol transmitting location information to Google to measure traffic density, is not easily possible. After installing a packet sniffer on a "rooted" Android smartphone, we notice that the data is transmitted encrypted to `https://www.google.com/loc/m/api` over TLS. In order to nevertheless be able to eavesdrop the data sent from our mobile phone, we perform a man-in-the-middle (MITM) attack, in which we insert ourselves into the communication between smartphone and Google. A Google Nexus S smartphone with Android 4.0.4 is used for all other investigations. The smartphone does not necessarily have to be "rooted".
With the help of the program mitmproxy one can realize a MITM attack [1]. If a connection from the smartphone to Google is established via TLS, mitmproxy forwards the request to Google but, instead of the Google certificate, returns a self-signed certificate to the smartphone. This allows mitmproxy to read the plaintext data. The proxy behaves in a transparent way and the Google servers cannot distinguish between communicating with the smartphone and with the proxy.
The Android system must accept the self-signed certificate, therefore, is has to be installed as a root certificate in the system. In contrast to older versions, Android version 4 allows subsequent installation of root certificates without much effort. The easy configuration of a system-wide proxy server to redirect the data over mitmproxy on mobile connections has been possible since Android 4.

The code for transmitting location information is not included in the open source code of Android 4. Since the current position of the smartphone is sent to Google, Google wants to prevent smartphone manufacturers from incorporating the functionality without regarding the Google privacy policy [15]. Google provides the code on request.
Luckily, the source code is available for older Android versions under the Apache License version 2.0 and is publicly available.

The protocol is a request/response protocol. In the request message, the smartphone sends Google status information of the GPS, wireless and mobile unit. On the one hand, the data amount depends on the units activated by the users, on the other hand on the configuration of the smartphone. If the smartphone, for example, does not have a GPS receiver and if Wi-Fi is enabled, only information about the wireless access points and radio cells in the surrounding area will be transferred. If Google is able to calculate the approximate location out of this data, the location is sent to the phone in the response message. A later explicit positioning by the user is accelerated, because although the positioning by GPS is more accurate, it can take several minutes without knowing the approximate location. Especially in urban environments, it is possible to locate the smartphone up to a few meters over Wi-Fi (without GPS) due to the high density of wireless networks and taking into account the signal strength.

Figure 1 shows a mitmproxy packet dump with location information sent to Google. The protocol is primarily based



00 02 00 a_0 ··· a_{ℓ_x} b_0 ··· b_7 "g" c_0 c_1 c_2 c_3 00
fixed
app name, app version, platform ID, distri. channel
cookie
encoding
MASF body length
plain
01 00 d_0 d_1 "g:loc/ql" 00 00 e_0 e_1 e_2 e_3 f_0 ··· f_{ℓ_f}
block
ID
service URI
service version
payload length
payload (protobuf)
zipped
01 01 d_0 d_1 "g:loc/ql" 00 00 6d 72 00 00 "ROOT" 00 e_0 e_1 e_2 e_3 "g" f_0 ··· f_{ℓ_f}
block
ID
service URI
service version
MIME
ROOT
payload length
g
payload (protobuf)

Fig. 2. MASF request message

on MASF (Mobile Application Sensing Framework). Figure 2 and 3 show a MASF message for request and response messages. Strings are UTF-8 encoded. Since the length of a string can be variable, it is stored in the first two bytes.
The request and response MASF header is described below. The payload of a message is encoded in the MASF Protocol Buffers format. A payload analysis is made in section II-C.

A. MASF Request Header

Figure 2 shows the message format for MASF request queries sent to Google. The first three bytes are constant, followed by a string giving the name of the app, the app version as well as the platform ID and the distribution channel. The values are separated by commas. A typical string we have seen is `location,-1,android,android,en_US`. Each MASF request header contains an eight-byte cookie that clearly identifies the phone. The encoding field always has the string "g" in our observations.

There are two types of request messages. In the case of a simple request message, the "block" field corresponds to the value $0x0100$. In this case, the payload data is not compressed. The payload data is only transmitted in a compressed way above a certain payload size. In this case the "block" field is set to $0x0101$.
In both types a unique ID is stored in the message. Messages can be temporarily correlated through this continuous sequence number. Apart from the length indications, the remaining fields are constants.

B. MASF Response Header

Figure 3 shows a MASF response message sent from the Google server to the smartphone. Each response message consists of either type "plain" ($0x8100$) or "multipart" ($0x8101$). In our analysis, we only observe the first type. We limit ourselves to this one. As in the case of the request message, each response message has an ID field in order to temporaily correlate the response messages. The status code indicates whether an error has occurred. In general, the value is $0x00c8 = 200$, which corresponds to "No error occurred". Depending on the packet size, the payload data is compressed. If the encoding field is the string "g", the payload data is compressed using gzip. If the payload data is not compressed, the field corresponds to an empty string.

C. Payload

```
message LatLngMsg
    required fixed32 Lat = 1;
    required fixed32 Lng = 2;
message LocationProfileMsg
    optional LatLngMsg LatLng = 1;
    optional int32 Accuracy = 3;
    optional int64 Timestamp = 6;
    optional int32 LocType = 8;
    optional int32 Altitude = 10;
    optional fixed32 Speed = 16;
    optional bool PluggedIn = 17;
message CellMsg
    required int32 Lac = 1;
    required int32 CellId = 2;
    optional int32 Mnc = 3;
    optional int32 Mcc = 4;
    optional int32 Rssi = 5;
    optional int32 RadioType = 10;
message WifiDeviceMsg
    required string MAC = 1;
    optional string SSID = 2;
    optional int32 Rssi = 4;
```

Fig. 4. Protocol Buffer template for request and response messages

Google uses Protocol Buffers (protobuf) to encode the payload data. Protocol Buffers is a data format developed by Google to serialize data structures [14]. In contrast to XML, Protocol Buffers in general can achieve a usually higher processing speed and data density because it is designed as a binary format. The protocol has been open source since July

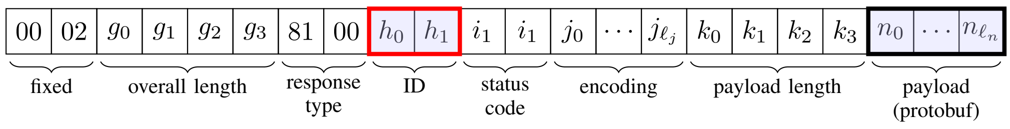


Fig. 3. MASF response message

```
message RequestMsg
    message PlatformProfileMsg
        required string Version = 1;
        optional string Platform = 2;
        optional string PlatformKey = 3;
        optional string Locale = 5;

        message CellularPlatformProfileMsg
            optional int32 RadioType = 1;
            optional string Carrier = 2;
            optional int32 HomeMnc = 4;
            optional int32 HomeMcc = 5;
        optional CellularPlatformProfileMsg CellPlatformProfile = 6;
    required PlatformProfileMsg PlatformProfile = 1;
    message RequestElementsMsg
        message CellularProfileMsg
            required CellMsg PrimaryCell = 1;
            required int64 Timestamp = 2;
        optional CellularProfileMsg CellularProfile = 1;

        message WifiProfileMsg
            required int64 Timestamp = 1;
            repeated WifiDeviceMsg WifiDevice = 2;
        optional WifiProfileMsg WifiProfile = 2;

        optional LocationProfileMsg LocationProfile = 3;
    repeated RequestElementsMsg RequestElements = 4;
```

Fig. 5. Protocol Buffers template for request messages

```
message ResponseMsg
    required int32 Status = 1;
    message LocReplyElementMsg
        required int32 Status = 1;
        optional LocationProfileMsg Location = 2;

        message DeviceLocationMsg
            optional LocationProfileMsg Location = 1;

            optional CellMsg Cell = 2;

            optional WifiDeviceMsg WifiDevice = 3;
        repeated DeviceLocationMsg DeviceLocation = 3;
    repeated LocReplyElementMsg LocReplyElement = 2;
    optional string PlatformKey = 3;
```

Fig. 6. Protocol Buffers template for response messages

2008 and is released under the 3-clause BSD license.
Figure 4 and 5 shows the Protocol Buffers template to encode the payload data for requests and figure 4 and 6 the one for response messages. The figures show only a selection of the most important elements for the sake of clarity. Each element has a key word, a "wire type" (data type), an identifier and an ID. The ID is specified after the equal sign and identifies the element in the data stream. The keywords `option` and `required` indicate whether the element is optional or mandatory. Elements with the keyword `repeated` can occur more than once or not at all. Elements of type `int32` or `int64` are represented as Varints. Varints are a method for encoding integer variables by one or more bytes in a space saving way.
Elements are grouped into messages (`messages`). However, an element can also be a message so that hierarchical structures can be constructed. Elements of type `string` represent UTF-8 encoded strings.

Figure 5 indicates that a request message may contain zero or more elements of type `RequestElementsMsg`. Each request element contains, depending on the smartphone features and the availability of location information, one or more of the three profiles `Cellular`, `Wifi` and `Location` (GPS). If, for example, there is no position data available from the GPS receiver, the message does not contain a `Location` profile.
Each request message contains a platform profile, which includes information about the phone model and the installed Android operating system (e.g. `platform => android/google/soju/crespo:4.0.4/IMM76D/ 299849:user/release-keys`). The Platform Key is a pseudonym to track the smartphone.
Google returns a status code and, if possible, the current geographical position of the smartphone. Optionally, the response message contains the geographic location of individual wireless access points and radio towers. Therefore, the user benefits from a faster determination of the position. A response message can contain a Platform Key, thus Google is able to change this key.

III. Waze Protocol

The Waze Protocol for transmitting location information is a simple request/response protocol such as the Google protocol. The complete source code is released under the GNU General Public License v2 and is freely available. In contrast to the Google protocol, position data is transmitted in the clear. We also use the program mitmproxy for recording the data stream. Since Waze ignores the global proxy settings, we use the app

$$\texttt{UID}, \underbrace{\texttt{628311428}}_{\text{server ID}}, \underbrace{\texttt{2Dyqtmg7r0HCZPFw}}_{\text{server cookie}}$$

$$\texttt{SeeMe}, \underbrace{\texttt{2}}_{\text{visability}}, \underbrace{\texttt{2}}_{\text{visability report}}, \underbrace{\texttt{T}}_{\text{download Wazers}}, \underbrace{\texttt{T}}_{\text{download reports}}, \underbrace{\texttt{T}}_{\text{download traffic}}, \underbrace{\texttt{1}}_{\text{allow ping}}, \underbrace{\texttt{-1}}_{\text{events radius}}$$

$$\texttt{SetMood}, \underbrace{\texttt{34}}_{\text{mood}}$$

$$\texttt{Location}, \underbrace{\texttt{9.946943}}_{\text{longitude}}, \underbrace{\texttt{53.569241}}_{\text{latitude}}$$

$$\texttt{At}, \underbrace{\texttt{9.951823}}_{\text{longitude}}, \underbrace{\texttt{53.561904}}_{\text{latitude}}, \underbrace{\texttt{0.000068}}_{\text{altitude}}, \underbrace{\texttt{-76}}_{\text{steering}}, \underbrace{\texttt{17}}_{\text{speed}}, \underbrace{\texttt{85068217}}_{\text{from node}}, \underbrace{\texttt{85067935}}_{\text{to node}}, \underbrace{\texttt{T}}_{\text{refresh users}}$$

$$\texttt{GPSPath}, \underbrace{\texttt{1334275968}}_{\text{GPS time}}, \underbrace{\texttt{3}}_{\text{count * 3}}, \underbrace{\texttt{9.965820}}_{\text{longitude}}, \underbrace{\texttt{53.569185}}_{\text{latitude}}, \underbrace{\texttt{57}}_{\text{altitude}}, \underbrace{\texttt{0}}_{\text{seconds gap}}$$

Fig. 7. Waze request message

ProxyDroid [16] on a "rooted" Android 4.0.4 smartphone to record the packets from the Waze app with mitmproxy.
The transmitted data is encoded as an ASCII string in the Waze protocol. A message can contain multiple blocks, which are introduced by keywords. A keyword is usually supplemented by parameters, which are separated by commas. Figure 7 shows an example of a data packet. The keywords are emphasized in blue. The number of blocks varies from message to message, figure 7 only shows an excerpt. The keywords `Location`, `GPSPath` and `UID` are important for our purposes. The current position of the smartphone is sent to Waze using the first keyword. `GPSPath` transmits a complete GPS path with several location points and `UID` transmits the credentials.
Waze very much follows a community approach. Therefore, the user usually registers himself before using the app. The transmission of login information such as user name and password is encrypted using TLS. If the user starts the app, the login information is transferred to the Waze server. The user gets a server ID and a cookie from the server. All subsequent messages sent to the Waze server contain the ID and the cookie, both of which are introduced by the keyword `UID`.

IV. Evaluation

We evaluate the Google and Waze protocol regarding privacy and authenticity. On the one hand, we deal with the question to which extent a user tracking is possible by Google and Waze. On the other hand, we investigate how an attacker can send false location information to Google or Waze to influence the traffic flow analysis.
As part of our investigations, we visualize the location information and find a serious flaw in the Network Location Provider Protocol (see section IV-C).

A. Privacy

To evaluate the tracking of both protocols, we drive a circular route. We do our measurements at night when there is little traffic. Therefore, a high average speed is reached. Wi-Fi is enabled and the smartphone is connected to an external GPS receiver via Bluetooth, which is necessary to have a stable GPS signal during the test drive.

Figure 8 shows the results if no user apps are running on the phone and only the Android system sends location data to Google. The driving direction is clockwise. The dashed line indicates the route driven. The distance between two lines indicates the speed (smaller distances suggest a lower speed). The blue and red curves are the results of two test runs. If the measuring points are connected, they are sent in one message.
In general, a high tracking accuracy can be noticed. When driving at higher speeds, larger gaps between two measurement points can be interpolated by knowing the route characteristics and the time stamps of the data points. The measurement points get closer together when we decrease the traveling speed, which increases the tracking accuracy.

If the Waze app runs in the background, the Waze data represents the GPS track almost accurately in figure 8. As in the case of the Google protocol, data points are collected and transferred as a bundle. The resolution is adjusted to the current speed. When driving at high speed, more measurement points are taken per time interval than when driving at a lower speed.

In another experiment, the GPS receiver of the smartphone is disabled so that the current position can only be determined via the surrounding access points and radio cells. This setup corresponds more to reality. The smartphone is often in the pants pocket. As a consequence, GPS reception is limited. Moreover, GPS is usually turned off to conserve battery life.
Figure 9 shows the visualization of the location information when GPS is not activated and no app is running. The red, dashed line indicates the covered distance and a circle with a time stamp marks a measurement point sent to Google. One can see that from 10:44 to 11:07 am and from 14:43 to 15:03

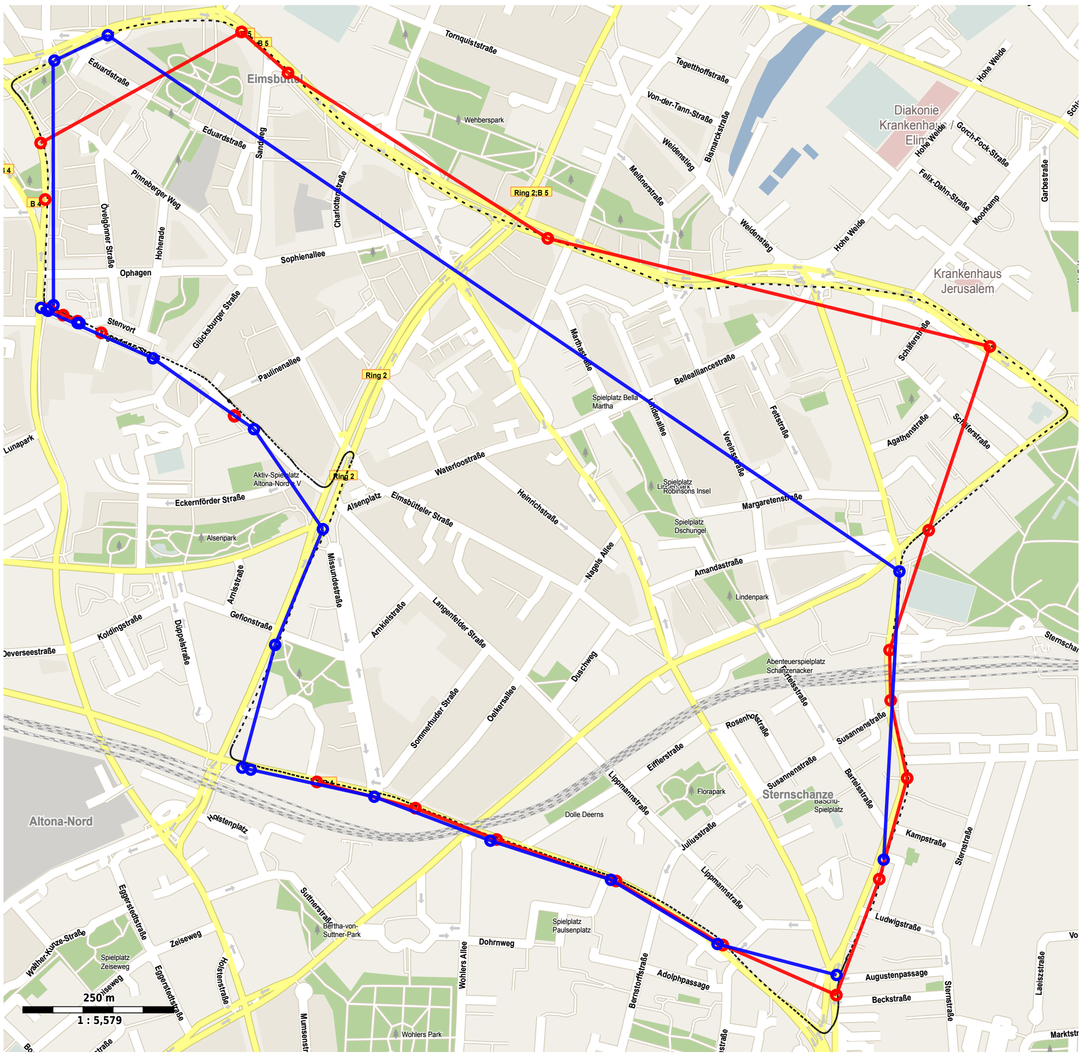


Fig. 8. Evaluation Google protocol [Wi-Fi on, GPS on], map data © OpenStreetMap contributors, CC BY-SA

pm the urban railway has been used (a large distance in a short time). In contrast, the distance between approximately 18:15 and 19:00 pm was covered very slowly (on foot). Due to the low speed, a better tracking is possible.

The figure shows that an Android phone without GPS receiver sends location information in intervals of 10 to 30 minutes. An accurate tracking of the user's movement is not possible. If the user remains at a fixed location for a long time, this can be detected. The recorded data shows that, for example, the phone was in a building of the Hamburg University of Technology from about 11:20 am to 2:30 pm. Between 3:10 and 5:50 pm the phone can be located at the University of Hamburg, and between 6 and 6:15 pm in the university's dining hall. Due to the high density of cellular and Wi-Fi towers, one can locate the smartphone even indoors up to a few meters.

The Waze app requires an activated GPS receiver but the current location is transmitted directly after starting the Waze app. User tracking is not possible.

If both Wi-Fi and GPS are turned off, there is no communication with the Google servers. The same also applies to Waze.

In order to track an individual user, Google must relate the messages to each other. We notice that each smartphone has a unique Platform Key and MASF cookie, both of which uniquely identify the phone. The Platform Key and cookie do not change even after rebooting the smartphone. The platform key is generated during the first start of the smartphone and the first communication with Google. The 8-byte cookie in the request MASF header is a random number. Once the cookie it set, it never changes.

However, even though Google Maps does not run in the background and is not even launched by the user, both Android and Google Maps send location data to Google. The packets captured by us either have the string `location,-1,android,android,en_US` or

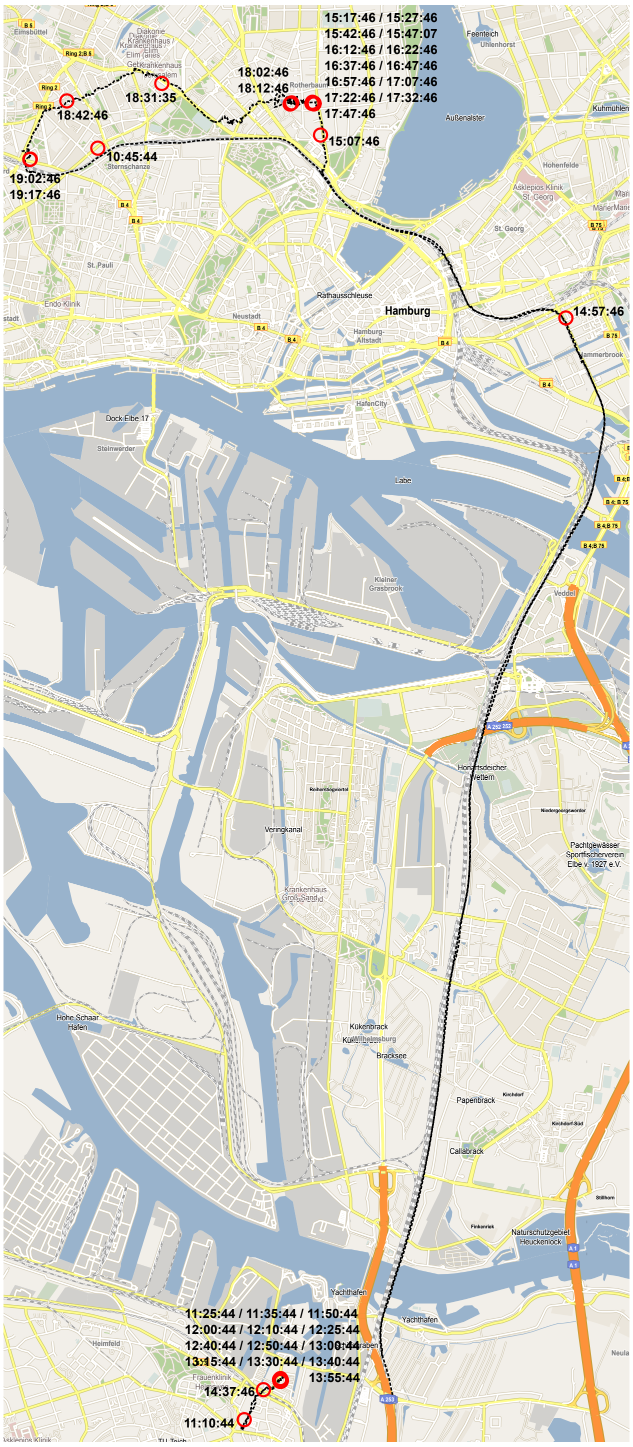


Fig. 9. Evaluation Google protocol [Wi-Fi on, GPS off], map data © OpenStreetMap contributors, CC BY-SA

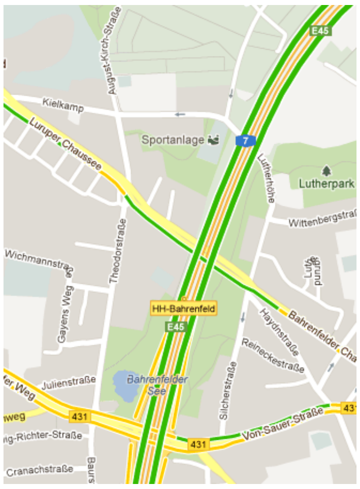
`location,1115,android,gmm,en_US` in the MASF request header (see string "a" in figure 2). The string `gmm` is the acronym for Google Maps Mobile. Both apps use different platform keys and cookies.
Every MASF message has a sequence number to put messages in time relation to each other. The sequence number is not required to track the smartphone because the measurement points usually contain time stamps.

The Waze app periodically sends messages to the Waze server. If a GPS receiver is available (and a valid location is found), a list of GPS coordinates, introduced by the keyword `GPSPath`, is transmitted to the Waze server. The keyword `location` is responsible for transmitting the current position of the smartphone to Google. This, for example, is done when the user starts the app.
If the user has logged in to Waze the position data can be associated with the user. Each message sent to Waze contains the unique server cookie and a server ID.

B. *Authenticity*

In the Google protocol, the TLS tunnel ensures data integrity so that it is impossible for an attacker to monitor a foreign phone or to modify data without being detected by Google. However, TLS is useless if the attacker controls the beginning of the TLS tunnel. Google cannot guarantee the authenticity of position data because a better authentication of smartphones is not desirable due to privacy reasons. The attacker can randomly select cookie and ID in the MASF header. A platform key is generated by Google. If no platform key is included in the request message, Google generates a new key and sends it back to the phone in the response message.

An attacker could send false location information to Google without being detected and therefore affect the traffic flow analysis. If, for example, an attacker drives a route and collects the data packets sent to Google, he can replay them later with a modified cookie, platform key and time stamps. The attack can be intensified by carrying out several delayed transmissions with different cookies and platform keys to simulate multiple cars. If the attacker adds noise to the measured values (e.g. to the signal strengths of wireless access points), uses different source IP addresses, a distinction between real and fake location information is no longer possible. Figure 10 exemplifies an attack in which an attacker creates an artificial traffic jam on highway ramp Hamburg-Bahrenfeld.
An attack in which the attacker has not previously driven the route can be realized since Google also accepts data from smartphones which do not have information of surrounding wireless access points. This is an important difference to the attack on TMC by Andrea Barisani and Daniele Bianco [3] in which the attacker can only take effect locally. In our attack, an attacker having reduced financial means can manipulate traffic data worldwide.


(a) Before the attack

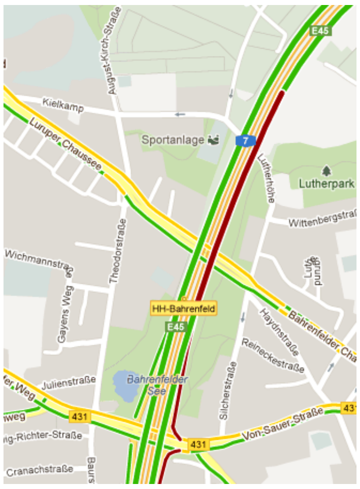

(b) Attack with wrong traffic data

Fig. 10. Highway ramp A7 - Hamburg-Bahrenfeld, map data © Google

As a consequence, an attacker can make people drive into traffic jams or keep roads clear if traffic data is used for navigation.

Such an attack scenario can also be applied to Waze. The attacker can send bogus position data to Waze and thus affect the navigation of other drivers. The attack becomes more difficult because the position data is associated with a user account. If the attacker simulates several vehicles, this requires several accounts with different e-mail addresses. However, we found out in our studies that position data can be transferred to Waze if user authentication is not done. Therefore, the attacker remains anonymous.

C. Network Location Provider Protocol

We use the Geolocation API in Google Gears to visualize the data points in figure 9. The signal strengths from surrounding wireless access points are mapped to geographic coordinates.
Google Gears is an extension for Web browsers, which makes it possible for web applications to use new features such as the Geolocation API. The browser submits, for example, data from an external GPS receiver, information about surrounding access points and radio towers to the network location provider. The network location provider calculates the current position (and optionally the address) from the data and sends it back to the browser. At the end of 2011, the Gears project was stopped as new browsers cover the functionality in HTML 5. The HTML 5 Geolocation protocol largely corresponds to the Network Location Provider Protocol, which is documented in [13]. Request and response messages are formatted in JSON.

We can figure out that, as in the case of the Google protocol, Google uses the same database with MAC addresses to calculate the geographical position. The database is continuously updated. If, for example, we put the access point somewhere else, Google "learns" the new position within a certain time.

In 2011, Samy Kamkar found out that access points can be located worldwide by using the Geolocation API [19]. Google responded to that and changed the system so that each request containing Wi-Fi data must at least have two MAC addresses of nearby access points.
In our studies we find out that it is still possible to locate a single access point[1]. If we send Google two MAC addresses, one of an access point whose geographic position we want to determine and a MAC address which is unknown to the system, Google will return the position of the access point in question.
This is a severe privacy issue. We observe that many people (unwittingly) publish the MAC address of their access point in the internet. Therefore, they can be located with an accuracy of a few meters. If the MAC address of the access point remains the same, a stalker can e.g. trace his victim even after the victim has moved to a new place. If a smartphone acts as an access point ("tethering") and the attacker knows the MAC address of the smartphone, it is theoretically possible to track the smartphone's owner (e.g. if the person stays longer abroad). In our tests, Google takes about ten days to add an unknown access point to its system.

V. Solution

In this section, we present an idea of a zero-knowledge protocol between a device $\mathcal{D}$ (e.g. a smartphone) and a provider $\mathcal{P}$ (e.g. Google), a protocol meeting the requirements of section I-A.
Zero-knowledge proofs were first introduced in 1985 by Goldwasser et. al [12] and continuously developed in recent

[1]In the meantime this bug has been fixed by Google. The MAC addresses of two nearby access points is necessary to return an accurate position.

decades. Our solution is based on a modified protocol from the paper "How to Win the Clone Wars: Efficient Periodic n-Times Anonymous Authentication" by Camenisch et al. [4].

The main idea of our protocol consists of linking location information with tickets. A device $\mathcal{D}$ once authenticates itself to its provider $\mathcal{P}$ (e.g. Google) and receives a so-called "ticket dispenser". With the help of this dispenser, $\mathcal{D}$ can generate tickets in order to send authenticated position data to $\mathcal{P}$. $\mathcal{P}$ is able to check the validity of the tickets, but can't link tickets to a specific device due to the zero-knowledge techniques used. Tracking of a single device is prevented.
Each ticket has a time stamp limiting its validity to a fixed time slot. This restricts the maximum number of valid data packets per time and device.

A. *Zero-Knowledge Proofs of Knowledge*

With the help of a proof of knowledge, a prover $\mathcal{P}$ can convince a verifier $\mathcal{V}$ of the fact that he knows a solution for a mathematical hard problem. On the one hand, an honest $\mathcal{P}$ can always convince $\mathcal{V}$ (*completeness* property); on the other hand, a dishonest $\mathcal{P}$ will fail to convince $\mathcal{V}$ with overwhelming probability if he attempts to cheat without knowing the correct solution (*soundness* property). A zero-knowledge proof of knowledge ZPK is a proof of knowledge where $\mathcal{V}$ obtains no further information from $\mathcal{P}$ other than the fact that $\mathcal{P}$ knows the solution of the underlying mathematically hard problem. This implies that executing the a zero-knowledge proof for the same secret twice, $\mathcal{V}$ cannot find out that the same secret has been used (*"unlinkability"*).
This paper follows the notation of Camenisch and Stadler [5] to describe zero-knowledge proofs of knowledge. E.g.:

$$\mathsf{ZPK}\left[(\omega) : x = g^{\omega}\ \right] \tag{1}$$

In ZPK (1) $\mathcal{P}$ proves to $\mathcal{V}$ the knowledge of the secret value w where w fulfils the relationship $x = g^{w}$. Secrets are marked in Greek characters. x and g are public values. Determining w by only knowing x and g is a mathematical hard problem in the groups given.

ZPKs are usually implemented by Σ-protocols, which are based on the Schnorr protocol [20], an interactive, challenge-response protocol. To achieve a high level of security the protocol has to be executed several times, which decreases protocol performance. The Fiat-Shamir heuristic [10] [17] and the random oracle model can be used to execute the protocol in a non-interactive way so that a ZPK becomes a signature proof of knowledge (SPK). Several predicates can be concatenated by logical AND/OR operators to model more complex proofs.

B. *Protocol*

The protocol consists of three sub-protocols.

1) Initialization: The provider $\mathcal{P}$ creates two cyclic groups $\mathbf{G}$ and $\mathbb{G}$ with generators $\langle \mathbf{g} \rangle = \langle \tilde{\mathbf{g}} \rangle = \langle \mathbf{h} \rangle = \mathbf{g}$ and $\langle g \rangle = \langle \tilde{g} \rangle = \langle h \rangle = \mathbb{G}$, respectively.
The group $\mathbf{G} = \mathbb{Z}_n^*$ is an RSA group of quadratic residues modulo $n = \mathbf{p} \cdot \mathbf{q}$ with two safe primes $\mathbf{p} = (2\mathbf{p}' + 1)$ and $\mathbf{q} = (2\mathbf{q}' + 1)$. The order of the group is $\mathbf{p}'\mathbf{q}'$ and unknown to $\mathcal{D}$. The provider proves in zero-knowledge that n is a special RSA modulus, and that the generators of the RSA group are quadratic residues modulo n [7].
The group $\mathbb{G} = \mathbb{Z}_p^*$ is a multiplicative group of order q. We choose the modulus p as a safe prime with $p = 2q + 1$.

The device $\mathcal{D}$ generates the secret key $sk_D \in_R \mathbb{Z}_q$ and calculates $pk_D = g^{sk_D} \bmod q$.
As a consequence, the public key of $\mathcal{P}$ is $(\mathbf{g}, \tilde{\mathbf{g}}, \mathbf{h}, g, \tilde{g}, h, p, q)$ and the one of $\mathcal{D}$ is pk_D. The secret key of $\mathcal{P}$ is the decomposition of the RSA modulus, i.e. $\mathbf{p}$ and $\mathbf{q}$.

2) Get Dispenser: The protocol consists of three steps (cf. figure 11).
First, device $\mathcal{D}$ authenticates itself to provider $\mathcal{P}$ to get a ticket dispenser. In the Google Live Traffic scenario, the authentication could be carried out through a Google account, for example. We assume that the following communication can be carried out over a secure and authenticated channel (e.g. SSL / TLS).
After the device has been authenticated, the dispenser function $f_{g,s}$ is generated. According to Dodi and Yampolskiy $f_{g,s}$ is a pseudo-random function [9] and defined as:

$$S = f_{g,s}^{DY}(c) = g^{\frac{1}{(s+c)}};\ s, c \in \mathbb{Z}_q^* \tag{2}$$

In the second step of the protocol, the initialization value s is negotiated without $\mathcal{P}$ knowing s. Therefore $\mathcal{D}$ generates the commitment C' containing the secret key sk_D and a random value s' and sends them back to $\mathcal{P}$. A SPK ensures that $\mathcal{D}$ really knows the values. The provider responds with a random value r' from which both parties calculate a commitment C.
In the last section of the protocol, $\mathcal{D}$ and $\mathcal{P}$ run the CL protocol [6]. With the help of the CL protocol $\mathcal{P}$ signs it secret key sk_D and the initialization value s of the pseudo-random function $f_{g,s}$ without knowing the two values. This step is important to prevent a later identification of $\mathcal{D}$ by $\mathcal{P}$.

3) Data Submission: (Encrypted) position data m can be only transmitted to the provider once in a time slot (e.g. every 15 minutes). Therefore we assign each 15-minute-interval to a timestamp c starting from a fixed point in time. If the starting point is, for example, the 01/01/13, then the time slot 0:00 to 00:15 am is assigned to value $c = 1$, the interval from 0:15 to 0:30 am is assigned to value $c = 2$, etc..

The device $\mathcal{D}$ calculates the actual ticket S from the actual timestamp c and the initial value s and proves in zero-knowledge the validity of both the ticket and the CL-signature from the "Get Dispenser" protocol (cf. figure 12).

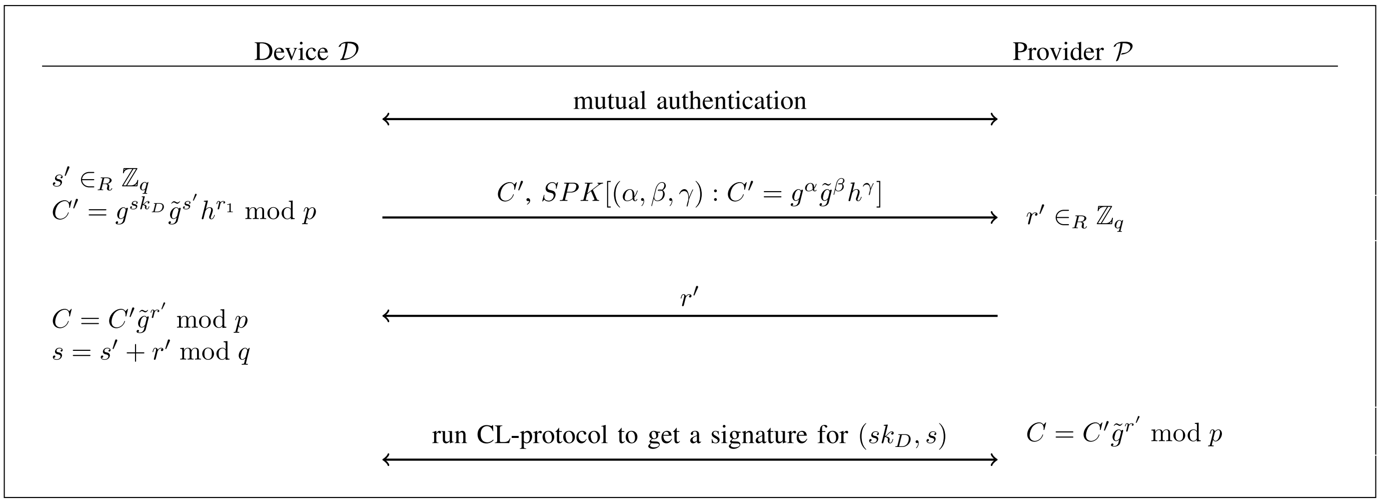

Device $\mathcal{D}$
Provider $\mathcal{P}$
mutual authentication
$s' \in_R \mathbb{Z}_q$
$C' = g^{sk_D} \tilde{g}^{s'} h^{r_1} \bmod p$
$C',\ SPK[(\alpha, \beta, \gamma) : C' = g^{\alpha} \tilde{g}^{\beta} h^{\gamma}]$
$r' \in_R \mathbb{Z}_q$
r'
$C = C' \tilde{g}^{r'} \bmod p$
$s = s' + r' \bmod q$
run CL-protocol to get a signature for (sk_D, s)
$C = C' \tilde{g}^{r'} \bmod p$

Fig. 11. Protocol run to get dispenser from $\mathcal{P}$

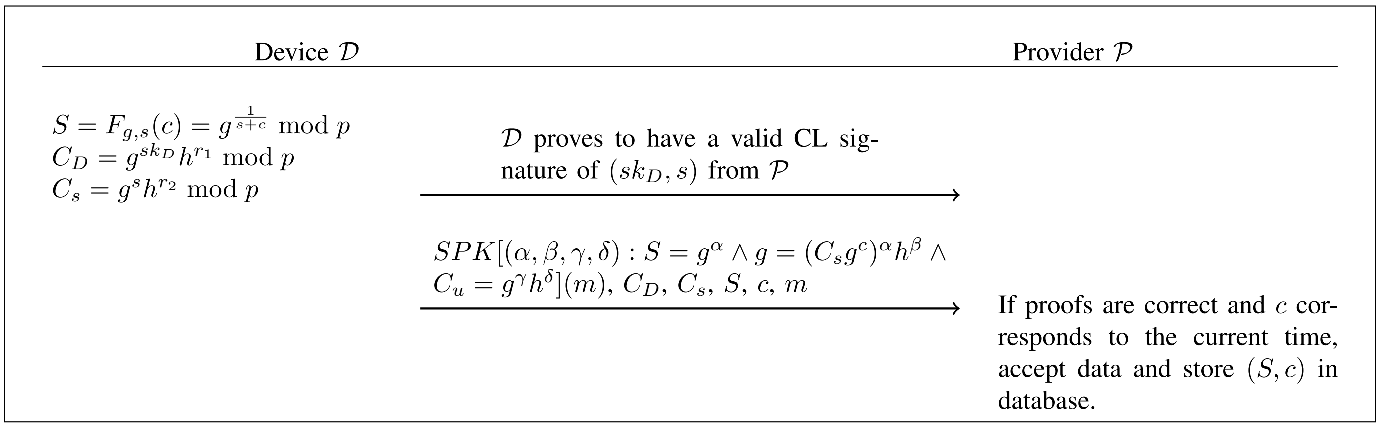

Device $\mathcal{D}$
Provider $\mathcal{P}$
$S = F_{g,s}(c) = g^{\frac{1}{s+c}} \bmod p$
$C_D = g^{sk_D} h^{r_1} \bmod p$
$C_s = g^{s} h^{r_2} \bmod p$
$\mathcal{D}$ proves to have a valid CL signature of (sk_D, s) from $\mathcal{P}$
$SPK[(\alpha, \beta, \gamma, \delta) : S = g^{\alpha} \wedge g = (C_s g^{c})^{\alpha} h^{\beta} \wedge C_u = g^{\gamma} h^{\delta}](m),\ C_D,\ C_s,\ S,\ c,\ m$
If proofs are correct and c corresponds to the current time, accept data and store (S, c) in database.

Fig. 12. Protocol run to submit position data from $\mathcal{D}$ to $\mathcal{P}$

If $\mathcal{D}$ behaves correctly, $\mathcal{P}$ accepts the position data m and stores (S, c) in a database. In the case of a new run of the protocol, $\mathcal{D}$ has to wait until the current time corresponds to the next timestamp c and is thus accepted by $\mathcal{P}$. If c is the same, the ticket S is the same. As a consequence, S and the potential attack is recognized. The zero-knowledge proofs prevent the counterfeiting of tickets. Owing to random components in the commitments and zero-knowledge proofs, protocol runs cannot be linked by $\mathcal{P}$ (*unlinkability*).

C. Performance

We have implemented our protocol on a server system (Intel Xeon X3460, @ 2.8 GHz, 8 GB RAM, x64) and an Android smartphone (Nexus S) to measure the performance. The bit length of the RSA modulus n and p is 1024bit. We use the MPIR 2.6.0 library to implement arithmetic operations. The results in table I show that the protocol is already feasible with standard hardware today.
It should be noted that our implementation still leaves room for improvement. In addition to general optimization techniques (e.g. look-up tables, specialized hardware, etc.), the computation can be distributed to multiple processor cores.
The smartphone calculations can mostly be pre-calculated in the background (e.g. when the phone is not currently in use). This can, for example, also happen while the smartphone is recharged to save battery.

D. Discussion

Even if, the protocol prevents the provider from linking protocol runs and tracking single devices, it is still possible that data packets can be linked by their IP address. It is important that after authentication (e.g. after executing the "Get-Dispenser" protocol) the smartphone waits for an IP address change before executing the "Submission" protocol.
In general, a mobile data connection is more often disconnected than a DSL connection. This is especially the case if the phone is moved and radio cells are changed. In addition, several providers (at least in Germany) automatically

TABLE I
PERFORMANCE COMPARISON

	Get Dispenser ($\mathcal{D}$)	**Get Dispenser** ($\mathcal{P}$)	**Submission** ($\mathcal{D}$)	**Submission** ($\mathcal{P}$)
Nexus S	112 ms	73 ms	318 ms	154 ms
Intel Xeon X3460	5 ms	3 ms	14 ms	7 ms

disconnect the connection after a few hours. If a connection is reestablished, the smartphone is usually assigned to a new IP. Furthermore, anonymity networks such as Tor [22] [21] can disguise the IP address. Thereby, it is possible to simultaneously achieve full privacy and authenticity.

The protocol can be extended. It is possible to identify users trying to send the provider data several times within the same time slot [4]. The validity of the ticket dispenser can be also limited. As a consequence, the device is forced to re-authenticate itself, for example every week. The accounts of attackers sending too many messages, can be removed from the system in the next run of the "Get-Dispenser" protocol.

VI. CONCLUSION

We have evaluated the Google and Waze protocol regarding privacy and authenticity. The anonymity of the user is not assured. In many cases a tracking is possible. In contrast, attackers can anonymously manipulate the traffic analysis and actively influence the navigation software. The attack can be carried out worldwide, requires no special equipment and is not overly expensive. The rapid development in the smartphone and automotive sector indicates the trend to consider real-time information for navigation. It is only a matter of time until hackers actively perform such an attack. We present a solution which increases the user's privacy and at the same time prevents attacks manipulating the traffic analysis.

REFERENCES

[1] Aldo Cortesi. *Mitmproxy 0.7 - an SSL-capable man-in-the-middle proxy*. 2012. URL: http://www.mitmproxy.org/.

[2] Julia Angwin and Jennifer Valentino-Devries. *Apple, Google Collect User Data*. Apr. 2011. URL: http://online.wsj.com/article/SB10001424052748703983704576277101723453610.html.

[3] Andrea Barisani and Daniele Bianco. "Injecting RDS-TMC Traffic Information Signals". In: *TELEMOBILITY 2007*. Nov. 2007.

[4] J. Camenisch, S. Hohenberger, M. Kohlweiss, A. Lysyanskaya, and M. Meyerovich. "How to win the clone wars: Efficient periodic n-times anonymous authentication". In: *ACM Conference on Computer and Communications Security. ACM*. 2006.

[5] J. Camenisch and M. Stadler. *Proof Systems for General Statements about Discrete Logarithms*. Tech. rep. 260. Institute for Theoretical Computer Science, ETH Zürich, 1997.

[6] Jan Camenisch and Anna Lysyanskaya. "A Signature Scheme with Efficient Protocols". In: *Security in Communication Networks*. Ed. by Stelvio Cimato, Giuseppe Persiano, and Clemente Galdi. Vol. 2576. Lecture Notes in Computer Science. 10.1007/3-540-36413-7_20. Springer Berlin / Heidelberg, 2003, pp. 268–289. URL: http://dx.doi.org/10.1007/3-540-36413-7%5C_20.

[7] Jan Camenisch and Markus Michels. "Proving in zero-knowledge that a number is the product of two safe primes". In: *Proceedings of the 17th international conference on Theory and application of cryptographic techniques*. EUROCRYPT'99. Prague, Czech Republic: Springer-Verlag, 1999, pp. 107–122. URL: http://dl.acm.org/citation.cfm?id=1756123.1756135.

[8] David Wang. *Stuck in traffic?* Feb. 2007. URL: http://googleblog.blogspot.de/2007/02/stuck-in-traffic.html.

[9] Yevgeniy Dodis and Aleksandr Yampolskiy. "A verifiable random function with short proofs and keys". In: *Proceedings of the 8th international conference on Theory and Practice in Public Key Cryptography*. PKC'05. Les Diablerets, Switzerland: Springer-Verlag, 2005, pp. 416–431. URL: http://dx.doi.org/10.1007/978-3-540-30580-4_28.

[10] Amos Fiat and Adi Shamir. "How to prove yourself: Practical solutions to identification and signature problems". In: *Advances in Cryptology – CRYPTO '86*. Ed. by A. M. Odlyzko. Vol. 263. Springer Verlag, 1987, pp. 186–194.

[11] Peter Fleischer. *Data collected by Google cars*. European Public Policy Blog. Apr. 2010. URL: http://googlepolicyeurope.blogspot.de/2010/04/data-collected-by-google-cars.html.

[12] S Goldwasser, S Micali, and C Rackoff. "The knowledge complexity of interactive proof-systems". In: *Proceedings of the seventeenth annual ACM symposium on Theory of computing*. STOC '85. Providence, Rhode Island, United States: ACM, 1985, pp. 291–304. URL: http://doi.acm.org/10.1145/22145.22178.

[13] Google. *gears - Geolocation API*. Nov. 2009. URL: http://code.google.com/p/gears/wiki/GeolocationAPI.

[14] Google. *Protocol Buffers - Google's data interchange format*. URL: http://code.google.com/p/protobuf/.

[15] Mike Lockwood. *masf and location provider support*. URL: https://groups.google.com/forum/?fromgroups#!topic/android-platform/c7z-JRK7_C4.

[16] Max Lv. *ProxyDroid 2.6.1*. URL: https://play.google.com/store/apps/details?id=org.proxydroid.

[17] D. Pointcheval and J. Stern. "Security proofs for signature schemes". In: *Advances in Cryptology – EUROCRYPT '96*. Ed. by U. Maurer. Vol. 1070. Springer Verlag, 1996, pp. 387–398.

[18] Roy Williams. *Youve got better things to do than wait in traffic*. Mar. 2011. URL: http://googlemobile.blogspot.de/2011/03/youve-got-better-things-to-do-than-wait.html.

[19] Samy Kamkar. *android map*. 2011. URL: http://samy.pl/androidmap/index.php.

[20] Claus-Peter Schnorr. "Efficient Signature Generation by Smart Cards". In: *Journal of Cryptology* 4.3 (1991), pp. 161–174.

[21] The Tor Project. *Orbot: Tor on Android*. 2013. URL: https://play.google.com/store/apps/details?id=org.torproject.android.

[22] The Tor Project. *Tor: anonymity online*. 2011. URL: https://www.torproject.org/.

[23] TISA. *The Traffic Message Channel (TMC)*. URL: http://www.tisa.org/technologies/tmc/.

[24] Waze. *Waze - Outsmarting Traffic, Together*. URL: http://www.waze.com/.