PayRide: Secure Transport e-Ticketing with Untrusted Smartphone Location

Marazzi Michele, Patrick Jattke, Jason Zibung and Kaveh Razavi

Computer Security Group, ETH Zurich

Abstract. The smartphone location is the basis for a plethora of popular applications, such as traffic navigation, games, and geotagging. Since the user can manipulate the reported location, it is possible to compromise these applications with fake locations. These attacks generally have a limited impact, but this is changing with the increasing level of trust in the smartphone location. As a prominent example, recent transport e-ticketing applications perform *financial transactions* based on the assumption that the smartphone location represents that of the user. Unfortunately, this assumption leads to location-based attacks with direct financial implications. We present FreeRide, a real-world attack that allows a malicious user to ride public transports for free. Existing mitigations against FreeRide are either ineffective or impractical since they attempt to enforce the integrity of the smartphone location. Instead of enforcing location integrity, our proposed mitigation, PayRide, establishes the user's location using the position of the public transport. We have formally verified the PayRide protocol and evaluated its boundary conditions based on a range of possible accuracies reported by the smartphone and public transport.

Keywords: Smartphone · Location · GPS · Transport · Attack.

1 Introduction

Virtually everyone with a smartphone today can report their geographic location to mobile applications. This has led to a viable market that companies have been quick to capitalize on. Examples range from navigation to online dating services. More recent ones include the mobile applications of public transport companies in several European countries that rely on the smartphone's location to provide automated e-ticketing [7,30]. These applications make the crucial assumption that the location reported by the smartphone is the same as the user's location. This assumption leads to practical location-spoofing attacks, which we demonstrate in this paper. To protect against these attacks, transport e-ticketing applications must preserve the link between the user and the smartphone despite inaccuracies and lack of trust in the location data provided by the smartphone. This paper presents PayRide, the first formally verified mitigation that preserves this link and protects such applications against location-spoofing attacks.

Smartphone location security. Mobile applications implicitly trust the provided interface for receiving location information. Thus, these applications are

exposed to Iago attacks [2], where either software or hardware manipulates the location information before it reaches the implicitly trusted interface. Recent Sybil attacks on Google Maps, for example, show that it is possible to fake traffic jams by spoofing the location of multiple emulated devices [5].

A straightforward approach to mitigate such attacks is to provide the smartphone with a mechanism to obtain a *proof of location*, either via anchor nodes [12, 15, 22] or from other smartphones in the vicinity [8, 19, 33]. The application can use the information from the nodes to attest the location provided by the smartphone. These mitigations are not deployed due to the cost of deploying a new infrastructure and the possibility of peer collusions. Another set of mitigations instead provides *location integrity* by ensuring that the GPS sensor data is signed by a Trusted Execution Environment (TEE) [9, 14, 25]. The application can then verify the integrity of the signed location information. Although location integrity is potentially interesting to mitigate known (low-severity) attacks, it does not provide sufficient protection in all scenarios of interest.

FreeRide. Recent transport e-ticketing applications make financial transactions based on the assumption that the location of the user is the same as the location reported by the smartphone. As we show in this paper, this extension of trust has dire security implications. In the context of malware, previous work has discussed that a malicious application could redirect requests intended for the TEE on a target device to another malicious device [3, 21]. We make a key observation that reversing this attack, i.e., using location information from another phone, enables location-spoofing attacks that bypass location integrity in the context of transport e-ticketing. We show a real-world instantiation of such attacks, which we call FreeRide, on the automated e-ticketing application of the Swiss public transport called SBB EasyRide. SBB EasyRide is designed to track the movement of a user based on the location provided by the smartphone and to charge the correct fare based on the distance traveled by different means of public transport. FreeRide enables free traveling across Switzerland while generating a valid ticket. The same applies to public transport of other European countries that rely on the exact same backend technology as SBB EasyRide [7].

PayRide. Given the increasing popularity of location-based e-ticketing applications and the severity of attacks such as FreeRide, developing a secure and deployable mitigation is of the utmost importance. Preserving the link between the user's actual location and the location reported by the smartphone is necessary for designing such a mitigation, which requires overcoming three main challenges. First, the location reported by the smartphone cannot be trusted. Second, the location information is inherently inaccurate, making it difficult to establish this link. Last, potentially unreliable Internet connections imply that this link cannot be preserved and verified in real-time.

We address these challenges in the design of PayRide, which is, to the best of our knowledge, the first solution to secure transport e-ticketing applications against location-spoofing attacks. To overcome the first challenge, PayRide relies on the transport location as a trusted anchor. This means that our second challenge must consider the location inaccuracies of both the smartphone user

and the transport. We formalize and calculate the minimum distances that can be charged based on these inaccuracies. Using these calculations, PayRide can then distinguish malicious from honest users without needing to trust the reported location. To address the last challenge, PayRide relies on a TEE to enforce location sampling and to preserve the order of the recorded information for calculating ticket prices or issuing fines in retrospect. Our evaluation of PayRide considers its boundary conditions based on the smartphone location accuracy, and our formal analysis proves its security.

Contributions. We make the following contributions:

1. We provide a security analysis of emerging transport e-ticketing applications, showing that existing mitigations cannot adequately protect against location-spoofing attacks.
2. To demonstrate the practical relevance of this threat, we build FreeRide, a real-world attack against the Swiss public transportation mobile application, SBB EasyRide.
3. We design and evaluate PayRide, a new mitigation against attacks such as FreeRide, and we formally verify the security of its protocol.

Responsible disclosure. We coordinated a responsible disclosure with SBB and were able to discuss FreeRide and its mitigation PayRide in a meeting with the company. SBB has assured us that effective measures have since been implemented to prevent the use of FreeRide. We would like to point out that using the public transport without a valid ticket in Switzerland has consequences under criminal law. During our experiments with FreeRide, we therefore always carried a valid ticket.

2 Background and Motivation

2.1 GPS on a Smartphone

A smartphone obtains its accurate location using a global navigation satellite system (GNSS). The acronym "GPS" refers to the GNSS used in the United States. Because of its widely accepted use, we will use GPS interchangeably with GNSS throughout this paper. GPS is based on multiple satellites orbiting around the Earth. Each GPS satellite continuously broadcasts a message to the Earth, which is then received by the device. The message contains a time and a position that allows the device to calculate its coordinates on Earth. Before the era of smartphones, GPS was exclusively used for its designed purpose: assisted navigation. GPS allows a user to pinpoint their location on a map in challenging environments such as deserts or oceans. Once smartphones became popular, applications like Google Maps made this service available to everyone. Further, the Internet connectivity of smartphones allows using the user's location in participatory sensing applications. For example, Google Maps links aggregated user locations to functional information such as car jams and crowded areas.

2.2 Current Attacks on the User's Location and Mitigations

Attacks on the user's location have focused on two scenarios: spoofing the GPS of the victim's device and poisoning participatory sensing databases [5,13,23,28,31,32]. In the first case, the victim is the user relying on their device to navigate. An attacker spoofing the GPS signal can hijack the victim's movement. In the second case, the attack victims are applications that rely on the user's location for participatory sensing. Researchers have shown that an attacker can poison a database by faking many identities, known as a Sybil attack [5]. In the context of the user's location, this translates to confusing services, such as making Google Maps report a car jam on a completely empty street.

Mitigations. To mitigate these attacks, a substantial amount of research describes how to generate a user's "proof of location". The majority of proof-of-location systems require deploying new infrastructures such as secure access points that attest a user's position [12,15,22]. However, none of them has been deployed due to the high cost of new infrastructures. A second category proposes to use nearby "witnessing" peers to verify a user's location [19,33]. This raises privacy concerns and requires a certain number of trusted peers nearby.

Trusted execution environment. Previous work suggests moving the GPS sensors reading to a restricted Trusted Execution Environment (TEE) that would sign the data and guarantee its integrity [9,14,25]. Even with the highest privileges, a user cannot read or modify the memory areas of the TEE. The user will access the TEE functionalities via specific API calls, typically used for cryptographic operations [16]. Recent work shows potential security problems of using TEEs as a trusted anchor [3,21]. In particular, if malware is deployed on a victim machine, the user might have their TEE requests hijacked to a maliciously controlled TEE. As we will show in this paper, variations of these issues also affect transport e-ticketing applications.

2.3 Transport e-Ticketing

An emerging class of applications tracks the user's location to provide automatic ticketing for public transports [26]. While traveling, the user can show an application-generated ticket, which is deemed valid by ticket controllers. Then, when the user has reached their destination, these applications analyze the journey to determine the traveled distance, and the user is billed accordingly. This mechanism was first introduced in Switzerland [26], and more recently, extended to other countries such as Austria [34] and Germany [6]. It is currently being implemented for Belgian [29] and French [7] public transports.

We show for the first time that the divergence from the intended GPS purpose results in direct financial benefits to malicious users, who can perform practical attacks on transport e-ticketing. This raises the following research question that we address in this paper: *can we achieve secure transport e-ticketing by relying on the untrusted location provided by the smartphone?*

3 Threat Model

We assume a victim transport e-ticketing application that relies on the smartphone location to provide services to the malicious smartphone user. In particular, the victim application generates a public transport ticket and bills the user at the end of their journey. The fare is derived from the user's collected location data. We assume that the malicious user wants to gain a direct financial benefit by providing the application with locations that do not correspond to their real physical positions. The aim of the attacker is to evade the detection of standard ticket control techniques that catch classic fraudulent behavior. In other words, we consider classical fraudulent behavior that would apply to standard tickets out of scope. This includes, for example, using public transport without any ticket, hoping not to get controlled. The application backend can rely on a TEE deployed on the user's smartphone, which is considered secure. We consider the user's GPS signal to be untrusted, which means that a malicious user can spoof and control their reported GPS location. We assume the transport company knows the location of their transport vehicles in real-time and that a controller can be present on the transport to perform checks on the user's ticket. These are standard assumptions for companies employing e-ticketing.

4 Challenges Overview

Our first challenge is to demonstrate that the critical assumption that the smartphone location is the same as the user location can be exploited by a malicious user targeting e-ticketing applications.

Challenge (C1). Demonstrating location-based attacks on real-world transport e-ticketing applications.

To solve this challenge, we describe the architecture of transport e-ticketing applications and the attacks against them, which we refer to as FreeRide (§ 5). Then, in § 6, we perform real-world FreeRide attacks on the Swiss public transport mobile application, *SBB Mobile*. Building these attacks required us to reverse-engineer parts of the application. We show the scalability of FreeRide by implementing location spoofing as a service, which significantly increases the impact of the attack by making it accessible to non-technical users.

Our next challenge is understanding whether any existing solution is practical and can secure transport e-ticketing applications.

Challenge (C2). Anaylsis of the security of existing proof-of-location proposals and their applicability to transport e-ticketing applications.

We address this challenge in § 7, where we present our security analysis of existing mitigations towards attacks such as FreeRide. We show that the existing proposals cannot protect e-ticketing applications or are severely impractical. Therefore, we consider the end-to-end deployment of a mitigation and study

the design requirements to make it practical and secure. Our last challenge is developing a mitigation based on the derived requirements.

Challenge (C3). Designing a secure mitigation against location-spoofing attacks for transport e-ticketing applications.

We address this challenge with the design of PayRide, our proposed mitigation discussed in § 8. PayRide preserves the link between the location of the smartphone and the user without needing to trust the position reported by the user. Designing PayRide requires answering the following fundamental questions: (i) How are GPS accuracies handled? (ii) How are communication delays addressed? (iii) Which impact do GPS accuracy and communication delays have on the application security? We provide a detailed analysis of PayRide boundary conditions based on the smartphone location accuracy and formally prove the security guarantees of its protocol. Because multiple European states share the same backend technology provider [7], FreeRide urges for the deployment of PayRide.

5 Functioning and Security of e-Ticketing Applications

We now describe the functioning of tracking-based e-ticketing applications.

Honest behavior. A user taking public transport starts the application and checks in to activate the tracking mode. From now on, their location is tracked as they travel, and a QR code representing a valid ticket is created. The ticket becomes immediately valid as the user can be controlled at any time during their journey. In case of a ticket inspection, the user shows the QR code, which is verified by the controller's application. After they reach their destination, the user stops the tracking mode and is billed according to the crossed geographical *zones* during their journey. Crossing more zones will result in a higher final ticket price. If the ticketing application does not detect that the user has traveled on public transport, the user is not billed [27].

Dishonest behavior. The public transports covered by e-ticketing applications do not employ physical barriers for checking the tickets. Instead, passengers traveling without a ticket are fined once a controller checks them. This is the same for both e-ticketing and paper-based ticketing users. To avoid fines, users must activate the tracking functionality *before* accessing the train and keep it active throughout the whole journey, as the user can be checked at any time. Users that have started the tracking mode *after* the controller entered the transportation are fined. In the next section, we will provide more details about the functioning of e-ticketing applications and reverse engineer SBB EasyRide.

Ticket applicability. Depending on the transport company and pricing model, the tracking ticket can be valid for multiple means of transportation. In the case of Swiss public transport, the same ticket is valid for all buses, trams, and trains. **For simplicity and without loss of generality, we consider trains as an example in the remainder of this paper.**

Fig. 1: **FreeRide$_{\text{local}}$.** A malicious user is on public transport. The application backend receives location updates corresponding to walking, providing the malicious user with a free but valid ticket.

5.1 Attacking Public Transport e-Ticketing

In our attack FreeRide$_{\text{local}}$, a malicious user makes use of the public transport service without paying for a ticket or getting fined. To this end, they exploit the public transport e-ticketing application, performing a local (on-device) location-spoofing attack. During the attack, the malicious user wants the application backend to receive user locations that do not correspond to a train ride. The locations can be crafted to appear as realistic movements, representing low-fare pathways or simulating low GPS accuracy with high communication delays. In Fig. 1, we report an example of FreeRide$_{\text{local}}$. In this scenario, the malicious user generates a walking pattern to bypass straightforward mitigations, such as invalidating a ticket corresponding to a user who is standing still. As we discuss in § 7, this simple attack could be mitigated with GPS integrity supported by TEE [14, 25]. However, slight variations bypass these mitigations, as we show next.

5.2 Bypassing Mitigations

We now discuss how an attacker can exploit the fact that *the location of the smartphone and the user are not necessarily the same* to bypass strong mitigations. Conceptually, this type of attack is similar to the *Cuckoo attack* originally commented in the context of malware [3, 21]. In a Cuckoo attack, malware installed on the victim device redirects requests that are originally intended for the local TEE to an attacker-controlled device. In this scenario, the victim cannot verify that they are interacting with their trusted device and not a malicious one. The owner of the device is thus the victim of the attack. In the context of wireless systems, this might also be interpreted as a *wormhole attack* [10].

FreeRide$_{\text{remote}}$. We make a key observation that "reversing" the Cuckoo attack, i.e., using location information from another phone, enables location-spoofing attacks that bypass location integrity in the context of transport e-ticketing. We refer to this attack as FreeRide$_{\text{remote}}$. This attack type applies to any scheme that aims to obtain location integrity via a TEE or similar technologies. This is because the backend cannot verify that it is communicating with the smartphone that the malicious user is *physically traveling with* in place of a different one.

In FreeRide$_{\text{remote}}$ (Fig. 2), two or more phones are used; one that the malicious user carries and others at different locations. For simplicity, in this example, we consider only one extra phone at a different location. Whenever requested, an honest user sends the position reported by the smartphone to the application. A malicious user, however, *replays* the decoy location provided by the second

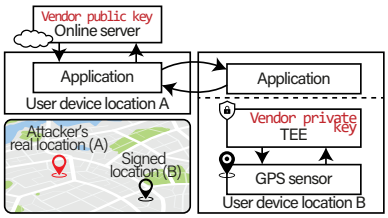

Fig. 2: **FreeRide$_{\text{remote}}$ attack scheme.** An online server requests the user's location. The malicious user forwards the request from the first to the second device. The second device is used to sign a different location with its TEE and sends it back.

phone (placed at a different location) to the application. Consequently, without any possible way to check the link between the location of the user and the smartphone, the application believes that the location of the user is the same as the (decoy) location of the second phone. Variations of this scenario are possible. For example, the phone is *not* required to be completely still. It could be held by a different user who does not take public transport, or it could be kept on local trains that remain in the same billing zone to travel a longer distance for a reduced fare. Further, the second device might contain the user private key.

Impact on mitigations. The impact of FreeRide$_{\text{remote}}$ is severe as it bypasses proposed mitigations based on location integrity [9]. These mitigations propose that a secure device (e.g., TEE) would sign the user location as proof of its integrity. However, the phone with the decoy location forwards a signed location to the phone carried by the malicious user. As such, the application cannot detect the ongoing attack. Unfortunately, of the previously proposed mitigations, location integrity is the only practically deployable solution, as we discuss in § 7. Our mitigation, discussed in § 8, assumes that the user location is untrusted and will not rely on location integrity to provide security.

6 FreeRide

We now describe our attacks performed on the e-ticketing application of the Swiss national railway company, SBB Mobile. First, we explain how Android applications obtain the user location (§ 6.1). Then, we obtain important insights by reverse engineering the SBB application (§ 6.2), and based on them, we execute both FreeRide$_{\text{local}}$ and FreeRide$_{\text{remote}}$ (§ 6.3). We conclude by demonstrating a Location-Spoofing-as-a-Service platform (§ 6.4). To the best of our knowledge, this is the first time in the literature that a location-based attack creates a *direct* financial advantage for the user.

6.1 Android Location API

To protect the users' privacy, applications that seek to obtain the smartphone location need to be granted specific permissions. The permissions are divided into (i) foreground, (ii) background, (iii) approximate, and (iv) precise location. The difference between foreground and background permissions is the allowed duration of the location access, while precise and approximate refers to the

location accuracy. Lastly, the application will further specify requirements such as the location update interval and priority, both affecting power consumption.

Fused location. The API provides a fused location, which is based on (i) power consumption, (ii) interface availability, and (iii) reported accuracy. GPS is preferred if network scanning is unavailable or if a higher precision is needed. Due to the nature of the transport e-ticketing applications, i.e., high precision and coverage, the location relies on the GPS sensor. Currently, to the best of our knowledge, there exists no TEE implementation of an integrity-protected GPS.

6.2 Application Reverse Engineering

We reverse engineer the SBB application (i.e., APK) using the dex-to-Java decompiler *jadx*. Using static analysis, we do not find any efforts to prevent the execution on a rooted device. To understand how the application works while the location data is used, we instrument it using the dynamic instrumentation framework Frida. EasyRide (the application tracking functionality) uses a rebranded version of the tracking technology by `fairtiq`, a service provider that generates automatic billing for various partners in Europe [7], including SBB. This is confirmed by the relevant EasyRide code being located in the `com.fairtiq.sdk` package. By dynamically instrumenting the application, we record its behavior during a normal trip and make the following four observations.

(O1) Tracking events. The application sends various events to the online server. The events describe the beginning and end of a ticket's validity, the device location, and the identified user activity. The application page specifies that the activity data is used to send a reminder to the user to perform a checkout [27].

(O2) Location provider. The application calls the fused location provider for the user position. To understand the type of request sent (see Section 6.1), we dynamically hook the necessary function call. We found that the location is requested with the "accurate" level and has an update interval of 2 seconds. With our mitigation (§ 8), we will describe how the update interval plays a major role in ensuring application security.

(O3) Location processing. The raw data location is transmitted to the backend without any preprocessing. In particular, the information sent to the server contains the latitude and longitude of the user position.

(O4) Messages batching. We found that data can be sent in batches and is flushed every 30 seconds. That means even though the location data is sampled frequently **(O2)**, it can be sent with large delays. As we will discuss (§ 8), reducing data delays plays an essential role in mitigating attacks.

Given our observations **(O1-O4)**, we conclude that only the location data is considered for the fare calculation. Next, we proceed with the design and evaluation of our FreeRide attacks.

6.3 Attacks Design and Evaluation

The attacker's goal is to deceive the SBB application to pretend that they are standing still or walking, while in reality, they are using public transport. Based

on our observations, we hypothesize that the ticket billing only depends on the GPS location. However, a limitation of the previous analysis is that we do not know how the data is processed in the backend. To better understand this, we exploit FREERIDE$_{\text{LOCAL}}$ by testing three different movement models of the spoofed location: (i) a fixed position (i.e., no movement), (ii) a simulated walk that does not cross stations, and (iii) a simulated walk that crosses stations.

FreeRide$_{\text{local}}$. In FREERIDE$_{\text{LOCAL}}$, the Android *location service* returns spoofed values to an API call in a completely transparent way for the victim application. To achieve this, we build an Android application that spoofs the user location and reproduces the three different types of movement. The application is designed as a mock location service, which is typically used to protect the user's privacy or to aid application development. The attack only requires changing the return value of the `isMock()` API, which we perform using the Smali Patcher tool. For our attack, we used a Pixel 5 running on Android 11, which we rooted using Magisk.

FreeRide$_{\text{remote}}$. Our second attack can bypass location integrity. As described before, we use two devices: one is static (the decoy), and the other one is traveling with the malicious user. Because there is currently no location integrity mechanism deployed in TEEs, we simulate it by keeping the decoy location fixed. For simplicity and without loss of generality, the decoy runs on an emulated Android. We provide more details about device emulation in Section 6.4. We generate a valid ticket on the decoy device, and we transmit it to the malicious user's phone. To perform the following attack evaluation, we use the latest unmodified version of the application installed from the Google Play Store.

Evaluation. We verified that in all three categories of spoofed movement (i.e., standing still, not crossing any stations, and crossing stations) with FREERIDE$_{\text{LOCAL}}$, as well as FREERIDE$_{\text{REMOTE}}$, the application always generates a ticket. The case in which the movement starts at a station and crosses a different one is the only occasion that results in the user being billed. This indicates that the attacker has complete control over the application by changing the location only. However, even if the application correctly generates a valid ticket, it could be that once a ticket controller checks it, some internal controls are applied to counteract fraud. To evaluate this, we performed several trips with different free tickets and were repeatedly checked by ticket controllers. To reduce our attack tests to the minimum necessary, we performed them on a limited distance that only crossed two zones and we always carried valid tickets as well. On all occasions, the free ticket was always considered to be valid by the controllers and never followed by any charges. Because the SBB application is used for all buses, trains, and trams, the attack allows a malicious user to travel for free across Switzerland.

6.4 LSaaS: Location-Spoofing-as-a-Service

We now demonstrate that it is practical for a malicious user to develop a *Location-Spoofing-as-a-Service* (LSaaS) platform, extending the attack to non-technical users. To this end, we designed an LSaaS platform for FREERIDE as

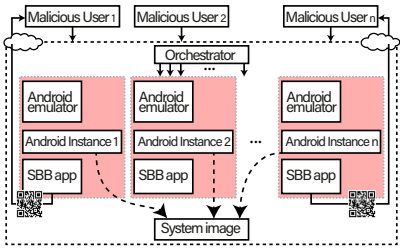

Fig. 3: **LSaaS architecture.** Malicious users connect to an online webpage and receive a valid and free ticket (QR code). Internally, the LSaaS orchestrator initializes the required copies of a virtualized Android environment. Each copy has a shared base system image that is prepared for the SBB application.

Table 1: **Comparison of location-spoofing efforts.** Of the practically-evaluated attacks, FreeRide is the first that provides a malicious user a direct financial benefit and is executed considering signed GPS data. It is also the first attack that scales to non-technical users via LSaaS.

Effort	Direct financial benefit	Practically evaluated	On-device spoofing	GPS-signal spoofing	FreeRide$_{\text{remote}}$ attack	LSaaS
FreeRide	●	●	●	○	●	●
GPS attacks [32]	○	●	○	●	○	○
Sybil attacks [5]	○	●	●	○	○	○
Hobbyist/gaming [20]	○	●	●	○	○	○

depicted in Fig. 3. We used Android Studio, which is bundled with a device emulator based on QEMU. The emulator runs an entire device, including GPS and networking, and the GPS location can be changed programmatically by using `adb`. Upon starting their ride, a malicious client obtains a valid ticket from the web server. First, the user provides account credentials for the SBB application. Then, the internal orchestrator creates and starts an ad-hoc Android emulator instance. The instance boots a preconfigured system with the SBB application, logs the user in, starts a new journey, and returns the QR code of the valid ticket to the user. The position remains fixed throughout the journey, resulting in no charge to the malicious user. We confirmed that the SBB application cannot detect that the source of the GPS location is fictitious, as the location service reports `false` for the `isMock()` API.

Scaling emulated devices. The orchestrator (Fig. 3) clones emulated devices on demand. To allow the LSaaS to scale to a large number of emulated devices, (i) a common system image is shared across all devices and (ii) a resource-optimized device configuration is used with a single core and a low-resolution screen.

In conclusion, by implementing a Location-Spoofing-as-a-Service, we showed that it is simple to scale our attack to a larger number of non-expert users. Table 1 provides a comparison of FreeRide with other known location-spoofing attacks. FreeRide is the first attack that provides a direct financial benefit for the attacker while bypassing location integrity. The results of FreeRide$_{\text{local}}$, FreeRide$_{\text{remote}}$ and LSaaS highlight the urgent need to develop and deploy a

practical and secure mitigation against FreeRide and other attacks based on bypassing location integrity.

7 Mitigation Requirements

We analyze proposed mitigations and evaluate their protection against the two variants of FreeRide. While we present previous work, we derive requirements that will guide us to the design of our mitigation PayRide (§ 8).

7.1 Existing Mitigations

In 2009, it was predicted that location would become a key aspect of mobile applications [24]. Since then, many researchers have focused on ways to generate proof of location for devices. However, the vast majority of these solutions are not practical, and as we will show, the few practical ones do not protect against FreeRide$_{\text{REMOTE}}$. There are three categories by which previous work can be classified into: (i) mitigations requiring a new infrastructure, (ii) mitigations involving peer interactions, and (iii) mitigations relying on signed GPS data.

Mitigations requiring a new infrastructure. These mitigations are based on deploying new infrastructure and require an access point in each area to be mapped [12, 15, 22]. An access point provides a marker for verifying that the user is present at a particular location. However, there are multiple limitations to this approach: (a) the deployment of such infrastructure does not scale with the expansion of the world roads, (b) without relying on GPS, the location accuracy may be strictly limited, and moreover, (c) applications would still be vulnerable to FreeRide$_{\text{REMOTE}}$. For these reasons, we do not consider this type of mitigation to be a viable solution. We argue that deploying a new infrastructure is a significant obstacle to the real-world adaption of a mitigation. Hence, we derive the following requirement for our mitigation:

> **Requirement (R1).** PayRide should be based on already existing and deployed infrastructure (i.e., GPS).

Mitigations involving peer interactions. This group of mitigations is based on a paradigm named *community-verified locations* [19, 33], in which the validation of a user's position is delegated to peers in their proximity. The approach (i) assumes that there are always multiple users at the same location, which might become difficult at low-traffic locations and during off-peak hours, (ii) requires that people are close enough to establish an ad-hoc network connection, (iii) may interrupt the user's connectivity as either Bluetooth or WiFi would need to be disconnected temporarily, (iv) requires a mechanism to identify malicious peers providing false location data, and (v) potentially compromises the users' privacy. Similar to the first class of mitigations, these proposals are impractical. As we consider the reliance on other users limiting and a potential security risk, we define the following requirement:

Requirement (R2). PayRide should provide application security without relying on any other users.

Mitigations relying on signed GPS data. Lastly, some work proposed signing the GPS location data using a TEE or any equivalent secure environment [9,14,25]. In these setups, device vendors embed a private key into the TEE memory during manufacturing. This key never leaves the device and is used to sign the GPS sensor values. After the online backend receives the signed user location, it verifies the validity of the signature using the device vendor's public key. To protect against replay attacks, the backend generates and transmits a random nonce to the device. This nonce must be included in the signed message and is incremented in each new position update.

Such a scheme cannot be used to protect against FreeRide$_{\text{remote}}$. First, the physical signal of the GPS can still be spoofed. Second, even if the source of the user location would be more secure than GPS, the TEE cannot validate the link between the user and device, which forms the basis of FreeRide$_{\text{remote}}$. Third, there is no way to prevent a malicious user from starting tracking directly on a different device. For the same reasons, the attack would also succeed if more complex schemes are employed, such as TEEs including the user's private key.

Mitigating FreeRide$_{\text{remote}}$ Attacks. Relying on FreeRide$_{\text{remote}}$, a malicious user appears to have a perfectly valid position outside or inside of the public transport network. Their position is valid, as it can be signed with a trusted key. This leads us to the third requirement for our mitigation:

Requirement (R3). PayRide should be able to match the physical location of the user to the location reported by the smartphone. However, it should assume the reported location to be untrusted.

7.2 The Ticket Controller

We now define the requirements for the ticket controller. There are multiple aspects to consider: first, relying on the ticket controller to visually inspect a reported user location creates a serious privacy concern caused by exposing the user's location history.

Second, differentiating between an honest and a malicious user might not be straightforward for the ticket controller. A malicious activity might be difficult to manually identify due to imprecise location data, network connection delays, or substantial variations in the user's movement speed. Lastly, the ticket validation should not require complex interactions for the ticket controller. We conclude with our last requirement:

Requirement (R4). PayRide should not rely on the ticket controller to check if the user is spoofing its location.

Given these four requirements, we now design our new mitigation PayRide. Table 2 summarizes the differences between the requirements in the design of PayRide and existing mitigations.

Table 2: **Comparison of proposed mitigations.** Proposals are either secure (✓) or vulnerable (✗) depending on the class of attack considered. Depending on their requirements, they can be practical (✓) or not (✗). No previous mitigation protects against FreeRide$_{\text{remote}}$ attacks.

Approach	On-device spoofing	GPS-signal spoofing	FreeRide$_{\text{remote}}$ attacks	Practical
PayRide	✓	✓	✓	✓
New infrastructure [15]	✓	✓	✗	✗
Peer interactions [8, 19, 33]	✓	✓	✗	✗
GPS signing [9]	✓	✗	✗	✓

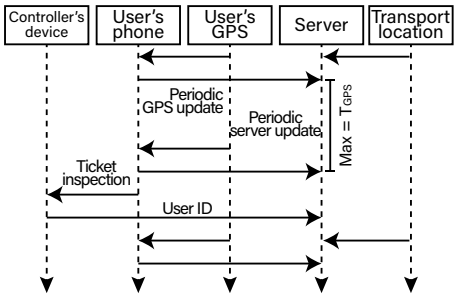

Fig. 4: **PayRide protocol.** The user periodically sends position updates to the application server. These updates must happen within a certain time limit. If a controller verifies a ticket, the user identifier is sent to the server. The server verifies that the user's location is within an acceptable range from the transport location.

8 PayRide

We now explain the principles of PayRide (§ 8.1), formalize its constraints (§ 8.2) and discuss how to safely define them (§ 8.3). Finally, we describe its formal verification model (§ 8.4) and conclude by describing the extension PayRide$_{\text{TEE}}$.

8.1 Operating Principles

With PayRide, a user who spoofs their location in a way that would give them financial benefit is equivalent to a user who does not possess a valid ticket. Both users are fined when their ticket is checked at any point along their journey. In the case of tracking ticket, the fine will be issued by the server backend. Following the requirements **R1-R2**, we have designed PayRide to rely exclusively on the reported GPS location. Instead of relying on location integrity, PayRide security is based on regular position updates sent by the user. While PayRide does not depend on trusting the sampled location (**R3**), TEE enables it to handle poor internet connectivity, as discussed in § 8.5 with PayRide$_{\text{TEE}}$.

PayRide protocol and ticket validation. We summarize the PayRide protocol in Fig. 4. The user periodically provides position updates to the server backend. These positions can correspond to the real user locations (GPS sensor) or carefully crafted pathways. If a maximum cumulative delay between updates (T_{GPS}) is not respected, the ticket is considered invalid. We relax this

requirement in Section 8.5. As standard practice of the affected parties, the ticket can be controlled one or multiple times at any given point in time during the journey. The controller behavior remains the same, scanning the user's ticket and verifying that it was created before they entered the train (**R4** and § 5). With PayRide, the user's identifier is sent to the application backend. In the backend, PayRide validates the user's location and considers that position for the fare calculation.

The user is fined if their reported location is not inside a defined margin of error from the transport location, considering the prior and next position, as reported by the user. A user moving faster than the train is considered malicious because users could misuse it to fake movements from the previous station to their current position during ticket inspections. In what follows, we formally describe the allowed error margin, the required timing for the location updates, and how they relate to PayRide security.

8.2 Formalizing Restrictions

PayRide should prevent two cases: (i) a user traveling for free (i.e., not crossing any stations) and (ii) a user paying less than they should (i.e., tracking movements across fewer zones). We consider fraudulent behaviors that apply to standard tickets (e.g., not having a ticket) to be out of scope.

Modeling the margins of error. To define a valid user position, we need to consider (i) the GPS margin of error and (ii) the update delay. The *GPS error margin* corresponds to the distance between the real physical location of the user and the one determined by the GPS (Fig. 5-❶-❸). We refer to the maximum allowed error as l_u for the user's position and as l_c for the transport position. The *update delay* represents the cumulative time between two consecutive location updates. This includes the time needed to calculate the GPS position and the time required for a network packet to travel to the backend server. Without loss of generality, we refer to the GPS update period (T_{GPS}) as the maximum time between position updates, measured at the time they arrive at the server (Fig. 5-❺). A longer delay results in an invalid ticket. We consider this maximum period T_{GPS} to be the same for user and transport location updates.

These margins need to be fixed for PayRide to correctly distinguish between an honest and a dishonest user. To be noted, l_c and T_{GPS} are not independent: if the allowed l_c is very small, it is difficult for the reported train position to match the user's position, even if they are close. This effect is worsened by a slow update rate (T_{GPS}) on a moving vehicle. If the update is not frequent enough, and the train is moving, the *reported* user's position can differ substantially from the train. However, PayRide needs to correctly identify an honest user, and hence, it is essential to consider the relationship between the locations reported by the user and the train. Note that l_u limits the shortest trip that PayRide can protect. For example, if l_u is larger than a train ride, a malicious user can spoof a static location throughout the entire trip, which would be correct upon ticket control. We calculate in § 8.3 the minimum distances that PayRide can protect.

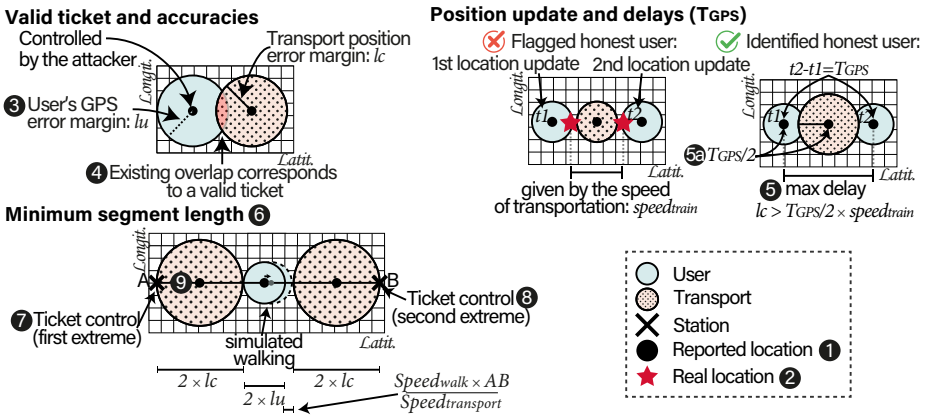

Fig. 5: **Maximum delay and GPS accuracies.** The user and train have a reported (❶) and a real (❷) position. The real position is within the radius l_u and l_c from the reported position (❸). The ticket is considered valid if there exists an intersection between the allowed error margin of the train and of the user's position (❹). The maximum accepted delay for a position update depends on the transportation speed and the train GPS error margin l_c (❺). The minimum segment length that can be secured (❻) depends on the allowed margins of error. For it to be secure, the user must never be able to generate a free-but-valid ticket, regardless of where they are controlled along the journey (❼,❽).

GPS update and l_c. The user's location needs to be updated frequently enough for an honest user to be correctly recognized during the ticket control. Because the train moves, the user's location is likely stale when compared in the backend with the train position. The distance between the user's real position (represented by the train position) and the stale position becomes smaller with a higher sampling rate. Given a certain T_{GPS}, we now describe the l_c boundaries to ensure honest users are always correctly identified.

For our analysis, we assume the worst-case scenario where the train position reaches the backend between two user location updates (Fig. 5-❺a). In other words, we consider that the user's location is stale by $\frac{T_{GPS}}{2}$. This is the worst condition, as PayRide considers the prior and next position reported by the user. Given the train movement of $speed_{train}$, the total distance is equal to $speed_{train} \times \frac{T_{GPS}}{2}$. Therefore, given the maximum speed and that both the user and train can report an inaccurate location, we require l_c to be higher than the following threshold to correctly identify an honest user:

$$l_c > \frac{T_{GPS}}{2} \times speed_{train} \tag{1}$$

8.3 Attack Based on the Error Margins

A malicious user can exploit location errors to falsely appear static (i.e., performing *distance fraud*). This allows them to travel for free if not controlled,

while holding a valid ticket in case they are. We now analyze the link between location errors (l_c and l_u) and the shortest journey that PayRide can secure.

Definitions. For a malicious user to possess a valid ticket, their reported position must always match the train reported position within the defined margins. This has to hold for the entire trip, as otherwise, there would exist a position along the route where the malicious user would be fined if checked. This is shown in the example in Fig. 5-❻, where a controller is checking tickets at the beginning of the trip (❼) or at the end (❽) on a train moving from station A to station B (length $\overline{AB}$). We remind the reader that a user who moves faster than a defined walking speed ($speed_w$) is billed as traveling.

Shortest secured journey. We now describe the attack that a malicious user can perform by exploiting l_c and l_u. After that, we derive the shortest journey of length $\overline{AB}$ such that malicious users cannot obtain a free but valid ticket.

As discussed, the malicious user aims to spoof their location such that (i) any ticket control along the journey $\overline{AB}$ is valid and (ii) their movement is never faster than walking speed, as otherwise, it would result in a fare. Intuitively, as shown in Fig. 5, the user can spoof their location to be in between A and B. This ensures that the error margin, $2 \times l_u$, covers as much as possible of their journey, increasing their chances of a matching position with the train. If the journey $\overline{AB}$ is shorter than $2 \times l_u$, a ticket where the user is standing still is always valid. We now extend the calculation by also considering the train location margin.

The first position where the user can be checked is at the beginning of the journey (❼). The real position of the train is A, but the reported position can be, in the worst case for the mitigation, distant by l_c (❾). This distance enhances the possible matching area of a malicious user. The last position where the user can be checked is at the end of the journey (❽) with the same possible error of l_c. Because the segment is contiguous, it follows that if the user has a valid position where both of these two extremes report a valid match, any position along the journey will be considered valid. Any non-linear connection between A and B can be reduced to a straight connection, which represents the worst-case for the mitigation. Therefore, the train margin of error adds $2 \times l_c$ matching coverage on the part of the segment *before* the user and adds $2 \times l_c$ matching coverage on the part of the segment *after* the user, for a total of $4 \times l_c$.

As a last step, we must consider that the user can generate a spoofed location that moves. The malicious user can further increase the covered length that matches with the train by slowly moving their position. The maximum coverage is obtained if they move at the maximum speed ($speed_w$) for the entire duration of the trip ($\frac{\overline{AB}}{speed_{train}}$). Summing all the contributions, we obtain the following maximum segment length that allows a malicious user to travel for free ($\overline{AB_{free}}$):

$$\overline{AB_{free}} = speed_w \times \frac{\overline{AB_{free}}}{speed_{train}} + 2 \times l_u + 4 \times l_c \approx 2 \times l_u + 4 \times l_c \qquad (2)$$

It follows that the condition on the segment length to achieve security is:

$$\overline{AB} > 2 \times l_u + 4 \times l_c \qquad (3)$$

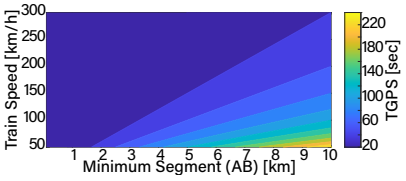

Fig. 6: **Location update period.** Relationship between ($\overline{AB}$), the train speed, and valid values for T_{GPS} (Equations (1) and (3)). The faster the train, the more frequent location updates are required.

Example of margins of error. We now provide example values for a train ride. We assume a high-speed transportation of up to 250 km/h with a minimum distance between stations of 5 km and a maximum walking speed of 5 km/h. As an example, we consider location accuracies of $l_u = l_c = 500$ m which satisfy Equation (2). Given Equation (1), this results in a T_{GPS} of roughly 14 seconds. Fig. 6 shows T_{GPS} for an $\overline{AB}$ in the range between 500 m and 10 km, and for a maximum train speed between 50 km/h and 300 km/h.

8.4 Tamarin Verification

We use Tamarin [17] to formally verify the correctness and security of PayRide. Tamarin is a symbolic formal verification tool widely used for protocol verification [1]. To this end, we define a formal Tamarin model of PayRide, based on the PayRide protocol and the described margins of error. In the model, two positions that are considered the same represent two locations that fall inside the allowed margins of error, which we defined before. We model the slowest possible (i.e., *worst*) GPS update period T_{GPS}. The user can behave in a malicious way (dishonest) by sending a *fake* location or behave correctly (honest) by sending correct location updates. The controller *might* check the user's ticket at any time. There is no restriction imposed on the user's behavior, meaning that we allow users to send a mix of honest and dishonest position updates. Lastly, we assume that the communication between the controller's device and the server is secure and that the train position is trusted.

Given the model, we prove that if the user is checked during a ride (i) a malicious user is always fined (i.e., no false negatives) while (ii) an honest user is never fined (i.e., no false positives). The formal model is simple (170 lines of code) and does not require implementing new cryptographic primitives on top of an already secured communication channel. The tool takes less than a minute to formally verify the model properties.

8.5 Handling Unreliable Internet Connections

So far, PayRide requires Internet connectivity to secure applications. A user with a loss of connectivity for more than T_{GPS} will incur a fine. This is an emerging condition due to Equation (1). Yet, it might be difficult to meet this in practice in areas with frequent loss of Internet connectivity. We now discuss how PayRide can be extended to use TEEs to remove this requirement.

We assume a TEE to be present, as they are available on most modern computing devices. As described earlier (§ 7.1), TEEs cannot be used to sign the integrity of the GPS signal to provide security. However, we consider the TEE by itself to be secure. For this reason, we use TEE not to protect integrity but to protect the **order** and **sampling period** of the positions that the client must send to the backend. We refer to our novel design as PayRide$_{\text{TEE}}$.

TEE to enforce timing. Previous work [9, 14, 25] employ TEEs to achieve integrity and preserve order of the sampled data. However, we make the observation that ordering alone does not entail *timing*. If a specific period for the location updates is not enforced, the application backend can receive a valid but outdated user location. For example, a transport e-ticketing application could receive trusted location updates from a walking user while the user is currently on a train.

PayRide$_{\text{TEE}}$. All the described principles of PayRide and our formalizations remain the same. However, when using a TEE, T_{GPS} represents the location *sampling* period. By using a secure interrupt, the TEE samples the location respecting T_{GPS} and signs it while including a counter to preserve the order. In the communication with the backend, a nonce is included to prevent replay attacks. In case of Internet connectivity loss, the signed updates are stored, and once the connectivity is restored, sent to the backend. To avoid attacks based on *stopping* location updates, the check-out operation is performed after the TEE has sent *all* position updates to the backend. In case the smartphone is powered off while the user is still checked in, the user is considered traveling without a valid ticket.

9 Related Work

We summarize research that focuses on (i) spoofing the GPS signal, (ii) poisoning participatory sensing applications, and (iii) spoofing the user's location.

Spoofing GPS signals. For many years, efforts have been made to demonstrate that GPS is not secure [13, 23, 31]. In these attacks, differently from FreeRide, the device owners are the victims of GPS spoofing. Typical attacks include jamming the GPS signal [28] or hijacking guided directions [32]. The main targets of GPS spoofing attacks have been unmanned aerial vehicles (UAVs), but they could also target people using driving assistants on the streets or the open sea.

Poisoning participatory sensing applications. Recently, researchers showed that participatory sensing applications are vulnerable to poisoning attacks [5], which is in line with previous work [11, 25]. Authors were able to artificially create areas detected as car traffic jams in Google Maps, making the application believe that a street is busy. Sybil attacks are a well-known threat for participatory applications and peer-to-peer systems [4, 18], in which malicious users influence the system functioning by using multiple identities.

Spoofing user locations. User-location spoofing has been used by hobbyists to unlock applications or features that are based on certain locations. For example,

games like Pokémon GO rely on the user's location to provide events to players and were subject to location spoofing attacks in the past.

10 Conclusions

We show that the assumption that the location of the smartphone matches that of the user has fundamental security implications for transport e-ticketing. We show-case this using a real-world attack, called FreeRide, that enables a malicious user to travel with a valid *free* ticket across Switzerland. Our analysis of existing mitigations shows that they are insufficient to protect against attacks such as FreeRide. To fill this gap, we designed PayRide, the first secure mitigation for transport e-ticketing applications. To make PayRide practical, we had to overcome a number of challenges, including relying on untrusted smartphone locations, GPS inaccuracies, and sporadic connectivity. We formally verified the PayRide protocol and evaluated its boundary conditions based on the smartphone location accuracy.

Acknowledgments. We thank our anonymous reviewers for their valuable feedback and Ralf Sasse for his help with Tamarin. This work was supported by the Swiss National Science Foundation under NCCR Automation, grant agreement 51NF40 180545, and the Swiss State Secretariat for Education, Research and Innovation under contract number MB22.00057 (ERC-StG PROMISE).

Disclosure of Interests. The authors have no competing interests to declare that are relevant to the content of this article.

References

1. Basin, D., Cremers, C., Dreier, J., Sasse, R.: Symbolically Analyzing Security Protocols Using TAMARIN **4**(4), in *ACM SIGLOG News*.
2. Checkoway, S., Shacham, H.: Iago Attacks: Why the System Call API Is a Bad Untrusted RPC Interface **41**(1), in *ACM SIGARCH Computer Architecture News*.
3. De Oliveira Nunes, I., Ding, X., Tsudik, G.: On the Root of Trust Identification Problem. In: ACM IPSN '21
4. Douceur, J.R.: The Sybil Attack. In: Springer IPTPS '02
5. Eryonucu, C., Papadimitratos, P.: Sybil-Based Attacks on Google Maps or How to Forge the Image of City Life. In: ACM WiSec '22
6. FAIRTIQ Ltd: Area of Validity, https://fairtiq.com/en/passengers/area-of-validity
7. FAIRTIQ Ltd: Partnership with FAIRTIQ, https://fairtiq.com/en/partner-with-fairtiq/public-transport-agencies
8. Gambs, S., Killijian, M.O., Roy, M., Traoré, M.: PROPS: A PRivacy-Preserving Location Proof System. In: IEEE SRDS '14
9. Hu, H., Chen, Q., Xu, J., Choi, B.: Assuring Spatio-Temporal Integrity on Mobile Devices with Minimum Location Disclosure **16**(11). https://doi.org/10.1109/TMC.2017.2683492, in *IEEE Transactions on Mobile Computing*.
10. Hu, Y.C., Perrig, A., Johnson, D.B.: Wormhole attacks in wireless networks. IEEE journal on selected areas in communications **24**(2), 370–380 (2006)
11. Huang, K.L., Kanhere, S.S., Hu, W.: Are You Contributing Trustworthy Data? The Case for a Reputation System in Participatory Sensing. In: ACM MSWiM '10

12. Javali, C., Revadigar, G., Rasmussen, K.B., Hu, W., Jha, S.: I Am Alice, I Was in Wonderland: Secure Location Proof Generation and Verification Protocol. In: IEEE LCN '16. https://doi.org/10.1109/LCN.2016.126
13. Larcom, J.A., Liu, H.: Modeling and Characterization of GPS Spoofing. In: IEEE HST '13
14. Liu, H., Saroiu, S., Wolman, A., Raj, H.: Software Abstractions for Trusted Sensors. In: ACM HotMobile '10. https://doi.org/10.1145/2307636.2307670
15. Luo, W., Hengartner, U.: Proving Your Location Without Giving up Your Privacy. In: ACM HotMobile '10. https://doi.org/10.1145/1734583.1734586
16. McGillion, B., Dettenborn, T., Nyman, T., Asokan, N.: Open-TEE: An Open Virtual Trusted Execution Environment. In: IEEE TrustCom '15. vol. 1
17. Meier, S., Schmidt, B., Cremers, C., Basin, D.: The TAMARIN Prover for the Symbolic Analysis of Security Protocols. In: Springer CAV '13
18. Newsome, J., Shi, E., Song, D., Perrig, A.: The Sybil Attack in Sensor Networks: Analysis & Defenses. In: IEEE IPSN '04
19. Nosouhi, M.R., Sood, K., Yu, S., Grobler, M., Zhang, J.: PASPORT: A Secure and Private Location Proof Generation and Verification Framework **7**(2), in *IEEE TCSS '20*.
20. Paay, J., Kjeldskov, J., Internicola, D., Thomasen, M.: Motivations and Practices for Cheating in Pokémon Go. In: ACM MobileHCI '18
21. Parno, B., McCune, J.M., Perrig, A.: Bootstrapping Trust in Modern Computers. Springer Science & Business Media (2011)
22. Pham, A., Huguenin, K., Bilogrevic, I., Dacosta, I., Hubaux, J.P.: SecureRun: Cheat-Proof and Private Summaries for Location-Based Activities **15**(8). https://doi.org/10.1109/TMC.2015.2483498, in *IEEE Transactions on Mobile Computing*.
23. Psiaki, M.L., Humphreys, T.E., Stauffer, B.: Attackers Can Spoof Navigation Signals Without Our Knowledge. Here's How to Fight Back GPS Lies **53**(8), in *IEEE Spectrum*.
24. Saroiu, S., Wolman, A.: Enabling New Mobile Applications with Location Proofs. In: ACM HotMobile '09. https://doi.org/10.1145/1514411.1514414
25. Saroiu, S., Wolman, A.: I Am a Sensor, and I Approve This Message. In: ACM HotMobile '10. https://doi.org/10.1145/1734583.1734593
26. SBB: EasyRide - the ticket that does things your way — SBB, https://www.sbb.ch/en/timetable/mobile-apps/sbb-mobile/easyride.html
27. SBB: Help with EasyRide
28. Setiadji, M.Y.B., Aji, B.P., Amiruddin, A.: Deceiving Smart Lock Trusted Place in Android Smartphones with Location Spoofing. In: IEEE ICOIACT '20
29. SNCB: Seamless Ticketing, https://www.belgiantrain.be/en/about-sncb/en-route-vers-mieux/innovation/seamless-ticketing
30. Swiss Federal Railways: The SBB online portal for trains and public transport — SBB, https://www.sbb.ch/en
31. Tippenhauer, N.O., Pöpper, C., Rasmussen, K.B., Capkun, S.: On the Requirements for Successful GPS Spoofing Attacks. In: ACM CCS '11
32. Zeng, K.C., Shu, Y., Liu, S., Dou, Y., Yang, Y.: A Practical GPS Location Spoofing Attack in Road Navigation Scenario. In: ACM HotMobile '17
33. Zhu, Z., Cao, G.: Toward Privacy Preserving and Collusion Resistance in a Location Proof Updating System **12**(1), in *IEEE Transactions on Mobile Computing*.
34. ÖBB Group: SimplyGo!, https://www.oebb.at/en/tickets-kundenkarten/online-mobile-ticketing/oebb-app/simplygo