VPriv: Protecting Privacy in Location-Based Vehicular Services

Raluca Ada Popa and Hari Balakrishnan
Massachusetts Institute of Technology
Email: {ralucap,hari}@mit.edu

Andrew J. Blumberg
Stanford University
Email: blumberg@math.stanford.edu

Abstract

A variety of location-based vehicular services are currently being woven into the national transportation infrastructure in many countries. These include usage- or congestion-based road pricing, traffic law enforcement, traffic monitoring, "pay-as-you-go" insurance, and vehicle safety systems. Although such applications promise clear benefits, there are significant potential violations of the *location privacy* of drivers under standard implementations (i.e., GPS monitoring of cars as they drive, surveillance cameras, and toll transponders).

In this paper, we develop and evaluate *VPriv*, a system that can be used by several such applications without violating the location privacy of drivers. The starting point is the observation that in many applications, some centralized server needs to compute a function of a user's *path*—a list of time-position tuples. VPriv provides two components: 1) the first practical protocol to compute path functions for various kinds of tolling, speed and delay estimation, and insurance calculations in a way that does not reveal anything more than the result of the function to the server, and 2) an out-of-band enforcement mechanism using random spot checks that allows the server and application to handle misbehaving users. Our implementation and experimental evaluation of VPriv shows that a modest infrastructure of a few multi-core PCs can easily serve 1 million cars. Using analysis and simulation based on real vehicular data collected over one year from the CarTel project's testbed of 27 taxis running in the Boston area, we demonstrate that VPriv is resistant to a range of possible attacks.

1 Introduction

Over the next few years, location-based vehicular services using a combination of in-car devices and roadside surveillance systems will become a standard feature of the transportation infrastructure in many countries. Already, there is a burgeoning array of applications of such technology, including electronic toll collection, automated traffic law enforcement, traffic statistic collection, insurance pricing using measured driving behavior, vehicle safety systems, and so on.

These services promise substantial improvements to the efficiency of the transportation network as well as to the daily experience of drivers. Electronic toll collection reduces bottlenecks at toll plazas, and more sophisticated forms of congestion tolling and usage pricing (e.g., the London congestion tolling system [14]) reduce traffic at peak times and generate revenue for transit improvements. Although the efficacy of automated traffic enforcement (e.g., stop-light cameras) is controversial, many municipalities are exploring the possibility it will improves compliance with traffic laws and reduces accidents. Rapid collection and analysis of traffic statistics can guide drivers to choose optimal routes and allows for rational analysis of the benefits of specific allocations of transportation investments. Some insurance companies are now testing or even deploying "pay-as-you-go" insurance programs in which insurance premiums are adjusted using information about driving behavior collected by GPS-equipped in-car devices.

Unfortunately, along with the tremendous promise of these services come very serious threats to the *location privacy* of drivers (see Section 3 for a precise definition). For instance, some current implementations of these services involve pervasive tracking—license-plate cameras, mandatory in-car GPS [16], toll

transponders, and insurance "black boxes" that monitor location and other driving information—with the data aggregated centrally by various government and corporate entities.

Furthermore, as a pragmatic matter, the widespread deployment and adoption of traffic monitoring is greatly impaired by public concern about privacy issues. A sizable impediment to further electronic tolling penetration in the San Francisco Bay Area is the refusal of a significant minority of drivers to install the devices due to privacy concerns [1]. Privacy worries also affect the willingness of drivers to participate in the collection of traffic statistics.

This paper proposes *VPriv*, a practical system to protect a user's locational privacy while efficiently supporting a range of location-based vehicular services. VPriv supports applications that compute functions over the *paths* traveled by individual cars. A path is simply a sequence of *points*, where each point has a random time-varying identifier, a timestamp, and a position. Usage-based tolling, delay and speed estimation, and pay-as-you-go calculations can all be computed given the paths of each driver.

VPriv has two components. The first component is an *efficient protocol for tolling and speed or delay estimation that does not compromise the location privacy of the drivers*. This protocol, which belongs to the general family of secure multi-party computations, guarantees that a joint computation between server and client can proceed correctly without revealing the private data of the parties involved. The result is that each driver (car) is guaranteed that no other information about its paths can be inferred from the computation, other than what is revealed by the result of the computed function. The idea of using multi-party secure computation in the vehicular setting is inspired from previous work [2, 3, 5]; however, these papers use multi-party computations as a black box, relying on general reductions from the literature. Unfortunately, these are extremely slow and complex, at least four orders of magnitude slower in our experiments than our implementation (see Section 8.3), which makes them unpractical.

Our main contribution here is the first *practically efficient* design, software implementation, and experimental evaluation of multi-party secure protocols for functions computed over driving paths. Our protocols exploit the specificity of cost functions over path time-location tuples: the path functions we are interested in consist of sums of costs of tuples, and we use homomorphic encryption [21] to allow the server to compute such sums using encrypted data.

The second component of VPriv addresses a significant concern: *making VPriv robust to attacks*. Although we can prove security against "cryptographic attacks" using the mathematical properties of our protocols, it is very difficult to protect against physical attacks in this fashion (e.g., drivers turning off their devices). However, one of the interesting aspects of the problem is that the embedding in a social and physical context provides a framework for discovering misbehavior. We propose and analyze a method using sporadic random spot-checks of vehicle locations that *are* linked to the actual identity of the driver. This scheme is general and independent of the function to be computed because it checks that *the argument* (driver paths) to the secure two-party protocol is highly likely to be correct. Our analysis shows that this goal can be achieved with a relatively small number of such checks, making this enforcement method inexpensive and minimally invasive.

We have implemented VPriv in C++ (and also Javascript for a browser-based demonstration). Our measurements show that the protocol runs in 100 seconds per car on a standard computer. We estimate that 30 cores of 2.4GHz speed, connected over a 100 Megabits/s link, can easily handle 1 million cars. Thus, the infrastructure required to handle an entire state's vehicular population is relatively modest. Our code is available at `http://cartel.csail.mit.edu/#vpriv`.

2 Related work

VPriv is inspired by recent work on designing cryptographic protocols for vehicular applications [2, 3, 5]. These works also discuss using random vehicle identifiers combined with secure multi-party computation

or zero-knowledge proofs to perform various vehicular computations. However, these papers employ multi-party computations as a black box, relying on general results from the literature for reducing arbitrary functions to secure protocols [24]. Such protocols tend to be very complex and slow. The state of the art "general purpose" compiler for secure function evaluation, Fairplay [29], produces implementations which run more than four orders of magnitude more slowly than the VPriv protocol, and scale very poorly with the number of participating drivers (see Section 8.3 for discussion of this point). Given present hardware constraints, general purpose solutions for implementing secure computations are simply not viable for this kind of application. A key contribution of this paper is to present a protocol for the *specific* class of cost functions on time-location pairs, which maintains privacy and are efficient enough to be run on practical devices and suitable for deployment.

Electronic tolling and public transit fare collection were some of the early application areas for anonymous electronic cash. Satisfactory solutions to certain classes of road-pricing problems (e.g., cordon-based tolling) can be developed using electronic cash algorithms in concert with anonymous credentials [8, 9, 18]. There has been a substantial amount of work on practical protocols for these problems that can run efficiently on small devices (e.g., [4]). Physical attacks based on the details of the implementation and the associated bureaucratic structures remain a persistent problem, however [10]. We explicitly attempt to address such attacks in VPriv. Our "spot check" methodology provides a novel approach to validating user participation in the cryptographic protocols, which we empirically verify is efficient. Furthermore, unlike VPriv, the electronic cash approach is significantly less suitable for more sophisticated road pricing applications, and does not apply at all to the broader class of vehicular location-based services such as "pay-as-you-go" insurance, automated traffic law enforcement, and aggregate traffic statistic collection.

There has also been a great deal of related work on protecting location privacy and anonymity while collecting vehicular data (e.g., traffic flow data) [33, 34, 35]. The focus of this work is different from ours, although they can be used in conjunction. They deal with potential privacy violations associated with the side channels present in anonymized location databases (e.g., they conclude that it is possible to infer to what driver some GPS traces belong in regions of low density).

Using spatial analogues of the notion of *k-anonymity* [6], some work focused on using a trusted server to spatially and temporally distort locational services [32, 36]. In addition, there has been a good deal of work on using a trusted server to distort or degrade data before releasing it. An interesting class of solutions to these problems were presented in the papers [31, 30], involving "cloaking" the data using spatial and temporal subsampling techniques. In addition, these papers [30, 31] developed tools to quantify the degree of mixing of cars on a road needed to assure anonymity (notably the "time to confusion" metric). However, these solutions treat a different problem than VPriv, because most of them assume a trusted server and a non-adversarial setting, in which the user and server do not deviate from the protocol, unlike in the case of tolling or law enforcement. Furthermore, for many of the protocols we are interested in, it is not always possible to provide time-location tuples for only a subset of the space.

Nonetheless, the work in these papers complements our protocol nicely. Since VPriv does produce an anonymized location database, the analysis in [30] about designing "path upload" points that adequately preserve privacy provides a method for placing tolling regions and "spot checks" which do not violate the location privacy of users. See Section 9 for further discussion of this point.

3 Model

In this section, we describe the framework underlying our scheme, our goals, and threat model. The framework captures a broad class of vehicular location-based services.

3.1 Framework

The participants in the system are *drivers*, *cars* and *a server*. Drivers operate cars, cars are equipped with transponders that transmit information to the server, and drivers also run *client software* which enacts the cryptographic protocol on their behalf.

For any given problem (tolling, traffic statistics estimation, insurance calculations, etc.), there is one logical server and many drivers with their cars. The server computes some function f for any given car; f takes the path of the car generated during an *interaction interval* as its argument. To compute f, the server must collect the set of points corresponding to the path traveled by the car during the desired interaction interval. Each point is a tuple with three fields: $\langle$`tag, time, location`$\rangle$.

While driving, each car's transponder generates a collection of such tuples and sends them to the server. The server computes f using the set of $\langle$`time,location`$\rangle$ pairs. If location privacy were not a concern, the `tag` could uniquely identify the car. In such a case, the server could aggregate all the tuples having the same tag and know the path of the car. In our case, these tags will be chosen at random so that they cannot be connected to an individual car. However, the driver's client application will give the server a *cryptographic commitment* to these tags (described in Sections 4.1, 5): in our protocol, this commitment binds the driver to the particular tags and hence the result of f (e.g., the tolling cost) without revealing the tags to the server.

We are interested in developing protocols that preserve location privacy for three important functions:

1. *Usage-based tolls:* The server assesses a path-dependent toll on the car. The toll is some function of the time and positions of the car, known to both the driver and server. For example, we might have a toll that sets a particular price per mile on any given road, changing that price with time of day. We call this form of tolling a *path toll*; VPriv also supports a *point toll*, where a toll is charged whenever a vehicle goes past a certain point.
2. *Automated speeding tickets:* The server detects violations of speed restrictions: for instance, did the car ever travel at greater than 65 MPH? More generally, the server may wish to detect violations of speed limits which vary across roads and are time-dependent.
3. *"Pay-as-you-go" insurance premiums:* The server computes a "safety score" based on the car's path to determine insurance premiums. Specifically, the server computes some function of the time, positions, speed, and acceleration of the car. For example, we might wish to assess higher premiums on cars that persistently drive close to the speed limit, or have frequent rapid acceleration events.

These applications can be treated as essentially similar examples of the basic problem of computing a localized cost function of the car's path represented as points. By localized we mean that the function can be decomposed as a sum of costs associated to a specific point or small number of specific points that are close together in space-time. This general framework can in fact be applied very broadly because of the general result that every polynomially-computable function has a secure multi-party protocol [12, 20]. However, as discussed in Section 8.3, these general results lead to impractical implementations: instead, we devise efficient protocols by exploiting the specific form of the cost functions.

In our model, each car's transponder obtains the point tuples as it drives and delivers them to the server. These tasks can be performed in several ways, depending on the infrastructure and resources available. For example, tuples can be generated as follows:

- A *GPS* device provides location and time, and the car's transponder prepares the tuples.
- *Roadside devices* sense passing cars, communicate with a car's transponder to receive a tag, and create a tuple by attaching time information and the fixed location of the roadside device.

Each car generates tuples periodically; depending on the specific application, either at random intervals (e.g., roughly every 30 seconds) or potentially based on location as well, for example at each intersection

if the car has GPS capability. The tuples can be delivered rapidly (e.g., via roadside devices, the cellular network, or available WiFi [28]) or they can be batched until the end of the day or of the month. Section 9 describes how to avoid leaking private information when transmitting such packets to the server.

Our protocol is independent of the way these tuples are created and sent to the server, requiring only that tuples need to reach the server before the function computation. This abstract model is flexible and covers many practical systems, including in-car device systems (such as CarTel [13]), toll transponder systems such as E-ZPass [19], and roadside surveillance systems.

3.2 Threat model

Many of the applications of VPriv are adversarial, in that both the driver and the operator of the server may have strong financial incentives to misbehave. VPriv is designed to resist five types of attacks:

1. The driver attempts to cheat by using a modified client application during the function computation protocol to change the result of the function.
2. The driver attempts to cheat physically, by having the car's transponder upload incorrect tuples (providing incorrect inputs to the function computation protocol):
   (a) The driver turns off or selectively disables the in-car transponder, so the car uploads no data or only a subset of the actual path data.
   (b) The transponder uploads synthetic data.
   (c) The transponder eavesdrops on another car and attempts to masquerade as that car.
3. The server guesses the path of the car from the uploaded tuples.
4. The server attempts to cheat during the function computation protocol to change the result of the function or obtain information about the path of the car.
5. Some intermediate router synthesizes false packets or systematically changes packets between the car's transponder and the server.

All these attacks are counteracted in our scheme as discussed in Section 9. Note however that in the main discussion of the protocol, for ease of exposition we treat the server as a passive adversary; we assume that the server attempts to violate the privacy of the driver by inferring private data but correctly implements the protocol. We believe this is a reasonable hypothesis since the server is likely to belong to an organization (e.g., the government or an insurance company) which is unlikely to engage in active attacks. However, as we discuss in Section 9, the protocol can be made resilient to a fully malicious server as well with very few modifications.

3.3 Design goals

We have the following goals for the protocol between the driver and the server, which allows the server to compute a function over a path.

Correctness. For the car C with path P_C, the server computes the correct value of $f(P_C)$.

Location privacy. We formalize our notion of location privacy in this paper as follows:

Definition 1 ***(Location privacy)*** *Let*

- $\mathbb{S}$ *denote the server's database consisting of* $\langle$*tag,time,location*$\rangle$ *tuples.*
- $\mathbb{S}'$ *denote the database generated from* $\mathbb{S}$ *by removing the tag associated to each tuple: for every tuple* $\langle$*tag, location, time*$\rangle \in \mathbb{S}$*, there is a tuple* $\langle$*location, time*$\rangle \in \mathbb{S}'$*.*
- C *be an arbitrary car.*

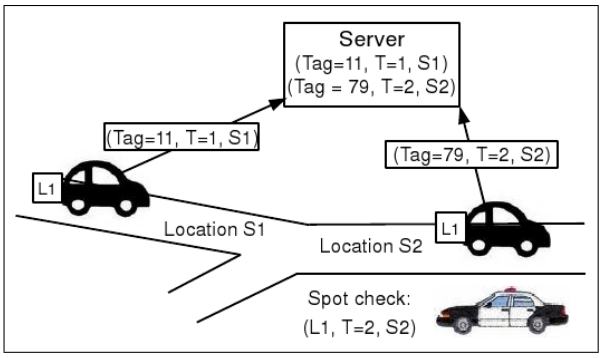

Figure 1: Driving phase overview: A car with license plate L1 is traveling from Location S1 at time 1 to Location S2 at time 2 when it undergoes a spot check. It uploads path tuples to the server.

- *$\mathcal{V}$ denote all the information available to the server in VPriv ("the server's view"). This comprises the information sent by C to the server while executing the protocol (including the result of the function computation) and any other information owned or computed by the server during the computation of f(path of C), (which includes $\mathbb{S}$).*
- *$\mathcal{V}'$ denote all the information contained in $\mathbb{S}'$, the result of applying f on C, and any other side information available to the server.*

The computation of f(path of C) preserves the locational privacy *of C if the server's information about C's tuples is insignificantly larger in $\mathcal{V}$ than in $\mathcal{V}'$.*

Here the "insignificant amount" refers to an amount of information that cannot be exploited by a computationally bounded machine. For instance, the encryption of a text typically offers some insignificant amount of information about the text. This notion can be formalized using simulators, as is standard for this kind of cryptographic guarantee. Such a mathematical definition and proof is left for an extended version of our paper.

Informally, this definition says that the privacy guarantees of VPriv are the same as those of a system in which the server stores only tag-free path points $\langle$`time,location`$\rangle$ without any identifying information. Note that this definition means that any side channels present in the raw data of $\mathbb{S}$ itself will remain in our protocols; for instance, if one somehow knows that only a single car drives on a certain roads at a particular time, then that car's privacy will be violated. See Section 9 for further discussion of this issue. Furthermore, observe that this definition requires that in general the tags be indistinguishable to the server from randomly drawn tags.

Efficiency. The protocol must be sufficiently efficient so as to be feasible to run on inexpensive in-car devices. This goal can be hard to achieve; modern cryptographic protocols can be computationally intensive.

4 Architecture

This section gives an overview of the VPriv system and its components. There are three software components: the client application, which runs on the client's computer, a transponder device attached to the car, and the server software attached to a tuple database. The only requirements on the transponder are that it store a list of random tags and generate tuples as described in Section 3.1. The client application is generally assumed to be executed on the driver's home computer or mobile device like a smart-phone.

The protocol consists of the following phases:

1. **Registration.** From time to time—say, upon renewing a car's registration or driver license—the driver must identify herself to the server by presenting a license or registration information. At that time, the client application generates a set of random tags that will be used in the protocol. We assume that these are indistinguishable from random by a computationally bounded adversary. The tags are

also transferred to the car's transponder, but *not* given to the server. The client application then cryptographically produces *commitments* to these random tags. We describe the details of computing these commitments in Sections 4.1 and 5. The client application will provide the ciphertext of the commitments to the server and these *will* be bound to the driver's identity; however, they do not reveal any information about the actual tags under cryptographic assumptions.

2. **Driving.** As the car is driven, the transponder gathers time-location tuples and uploads them to the server. Each path tuple is unique because the random tag is never reused (or reused only in a precisely constrained fashion, see Section 5). The server *does not know* which car uploaded a certain tuple. To ensure that the transponder abides by the protocol, VPriv also uses sporadic random spot checks that observe the physical locations of cars, as described in Section 7. At a high level, this process generates tuples consisting of the actual license plate number, time, and location of observation. Since these spot checks record license plate information, the server knows which car they belong to. During the next phase, the client application will have to prove that the tuples uploaded by the car's transponder are consistent with these spot checks.
3. **Reconciliation.** This stage happens at the end of each interaction interval (e.g., at the end of the month, when a driver pays a tolling bill) and computes the function f. The client authenticates itself via a web connection to the server. (It may also anonymously upload any tuples that are missing at the server.) If the car had undergone a spot check, the client application has to prove that the tuples uploaded are consistent with the spot checks before proceeding (as explained in Section 7). Then, the client application initiates the function computation. The server has received tuples from the driver's car, generated in the driving phase. However, the server has also received similar tuples from many other cars and *does not know* which ones belong to a specific car. Based on this server database of tuples as well as the driver's commitment information from registration, the server and the client application conduct a cryptographic protocol in which:

    - The client computes the desired function on the car's path, the path being the *private input*.
    - Using a zero-knowledge proof, the client application proves to the server that the result of the function is correct, by answering correctly a series of challenges posed by the server *without revealing the driver's tags*.

This framework is analogous to the setup of [2, 3]. To implement this protocol, VPriv uses a set of modern cryptographic tools: a homomorphic commitment scheme and random function families. We provide a brief overview of these tools below. The experienced reader may skip to Section 5, where we provide efficient realizations that exploit details of our restricted problem setting.

4.1 Overview of cryptographic mechanisms

A **commitment scheme** [27] consists of two algorithms, COMMIT and REVEAL. Assume that Alice wants to commit to a value v to Bob. In general terms, Alice wants to provide a ciphertext to Bob from which he cannot gain any information about v. However, Alice needs to be bound to the value of v. This means that, later when she wants to reveal v to Bob, she cannot provide a different value, $v' \neq v$, which matches the same ciphertext. Specifically, she computes $\text{COMMIT}(v) \rightarrow (c, d)$, where c is the resulting ciphertext and d is a decommitment key with the following properties:

- Bob cannot feasibly gain any information from c.
- Alice cannot feasibly provide $v' \neq v$ such that $\text{COMMIT}(v') \rightarrow (c, d')$, for some d'.

We say that Alice reveals v to Bob if she provides v and $d(v)$ to Bob, who already holds $c(v)$.

We use a **homomorphic commitment** scheme due to Pedersen [21], in which performing some operation on the ciphertexts corresponds to some (other or the same) operation on the plaintext. For instance, a

$COST$	Path tolling cost computed by the client and reported to the server.
$c(x), d(x)$	The **c**iphertext and **d**ecommitment value resulting from committing to value x. That is, $\text{COMMIT}(x) = (c(x), d(x))$.
v_i	The random tags used by the **v**ehicle's transponder. A subset of these will be used while driving.
(s_i, t_i)	A pair of a random tag uploaded at the **s**erver and the **t**oll cost the server associates with it. $\{s_i\}$ is the set of all random tags the server received within a tolling period with $t_i > 0$.

Figure 2: Notation.

commitment scheme that has the property that $c(v) \cdot c(v') = c(v + v')$ is homomorphic. Here, the decommitment key of the sum of the plaintexts is the sum of the decommitment keys $d(v + v') = d(v) + d(v')$.

A **secure multi-party computation** [20] is a protocol in which several parties hold private data and engage in a protocol in which they compute the result of a function on their private data. At the end of the protocol, the correct result is obtained and none of the participants can learn the private information of any other beyond what can be inferred from the result of the function. In this paper, we designed a variant of a secure two-party protocol. One party is the car/driver whose private data is the driving path, and the other is the server, which has no private data. A **zero-knowledge proof** [12] is a related concept that involves proving the truth of a statement without revealing any other information.

A **pseudorandom function family**[22] is a collection of functions $\{f_k\} : D \to R$ with domain D and range R, indexed by k. If one chooses k at random, for all $v \in D$, $f_k(v)$ can be computed efficiently (that is, in polynomial time) and f_k is indistinguishable from a function with random output for each input.

5 Protocols

This section presents a detailed description of the specific interactive protocol for our applications, making precise the preceding informal description. For concreteness, we describe the protocol first in the case of the tolling application; the minor variations necessary to implement the speeding ticket and insurance premium applications are presented subsequently.

5.1 Tolling protocol

We first introduce the notation in Table 2. For clarity, we present the protocol in a schematic manner in Figure 3. This protocol is a case of two party-secure computation (the car is a malicious party with private data and the server is an honest but curious party) that takes the form of zero-knowledge proof: the car first computes the tolling cost and then it proves to the server that the result is correct. Intuitively, the idea of the protocol is that the client provides the server an encrypted version of her tags on which the server can compute the tolling cost in ciphertext. The server has a way of verifying that the ciphertext provided by the client is correct. The privacy property comes from the fact that the server can perform only one of the two operations at the same time: either check that the ciphertext is computed correctly, or compute the tolling cost on the vehicle tags using the ciphertext.

These verifications and computations occur within a round, and there are multiple rounds. The server has a probability of at least $1/2$ to detect whether the client provided an incorrect COST, as argued in the proof below. The round protocol should be repeated s times, until the server has enough confidence in the correctness of the function. After s rounds, the probability of detecting a misbehaving client is at least $1 - (1/2)^s$, which decreases exponentially. Thus, for $s = 10$, the client is detected with 99.9% probability.

1. **Registration phase:**
   (a) Each client chooses random vehicle tags, v_i, and a random function, f_k, by choosing k at random.
   (b) Encrypts the selected vehicle tags by computing $f_k(v_i), \forall i$, commits to the random function by computing $c(k)$, commits to the encrypted vehicle tags by computing $c(f_k(v_i))$, and stores the associated decommitment keys, $(d(k), d(f_k(v_i)))$.
   (c) Send $c(k)$ and $c(f_k(v_i)), \forall i$ to the server. This will prevent the car from using different tags.
2. **Driving phase:** The car produces path tuples using the random tags, v_i, and sends them to the server.
3. **Reconciliation phase:**
   (a) The server computes the associated tolling cost, t_j, for each random tag s_j received at the server in the last period based on the location and time where it was observed and sends (s_j, t_j) to the client only if $t_j > 0$.
   (b) The client computes the tolling cost $COST = \sum_{v_i = s_j} t_j$ and sends it to the server.
   (c) The **round protocol** begins:

Client	Server
(i) Shuffle at random the pairs (s_j, t_j) obtained from the server. Encrypt s_j according to the chosen f_k random function by computing $f_k(s_j), \forall j$. Compute $c(t_j)$ and store the associated decommitments.	
Send to server $f_k(s_j)$ and $c(t_j)$, $\forall j$ $\rightarrow$	
	(ii) The server picks a bit b at random. If $b = 0$, challenge the client to verify that the ciphertext provided is correct; else, challenge the client to verify that the total cost based on the received ciphertext matches $COST$.
	$\leftarrow$ Challenge random bit b
(iii) If $b = 0$, the client sends k and the set of (s_j, t_j) in the shuffled order to the server and proves that these are the values she committed to in step (i) by providing $d(k)$ and $d(t_j)$. If $b = 1$, the client sends the ciphertexts of v_i ($\{f_k(v_i)\}$) and proves that these are the values she committed to during registration by providing $d(f_k(v_i))$. The client also computes the intersection of her and the server's tags, $I = \{v_i\} \cap \{s_j\}$. Let $T = \{t_j : s_j \in I\}$ be the set of associated tolls to s_j in the intersection. Note that $\sum_T t_j$ represents the total tolling cost the client has to pay. By the homomorphic property discussed in Section 4.1, the product of the commitments to these tolls t_j, $\prod_{t_j \in T} c(t_j)$, is a ciphertext of the total tolling cost whose decommitment key is $D = \sum_{t_j \in T} d(t_j)$. The server will compute the sum of these costs in ciphertext in order to verify that $COST$ is correct; the client needs to provide D for this verification.	
If $b = 0$, $d(k), d(t_i)$ else $D, d(f_k(v_i))$ $\rightarrow$	
	(iv) If $b = 0$, the server verifies that all pairs (s_j, t_j) have been correctly shuffled, encrypted with f_k, and committed. This verifies that the client computed the ciphertext correctly. If $b = 1$, the server computes $\prod_{j:\exists\ i,\ f_k(v_i) = f_k(s_j)} c(t_j)$. As discussed, this yields a ciphertext of the total tolling cost and the server verifies if it is a commitment to $COST$ using D. If all checks succeed, the server *accepts* the tolling cost, else it *denies* it.

Figure 3: VPriv's protocol for computing the path tolling cost (small modifications of this basic protocol work for the other applications). The arrows indicate data flow.

The number of rounds is fixed and during registration the client selects a pseudorandom function f_k and provides a set of commitments for each round.

Note that this protocol also reveals the number of tolling tuples of the car because the server knows the size of the intersection (i.e. the number of matching encryptions $f_k(v_i) = f_k(s_j)$ in iv) for $b = 1$). This may be a minor issue, but there is an easy fix: Before the reconciliation phase, the motorist uploads several junk tuples that consist of a valid tag and a junk location and time. The server sends such tuples to the client during step 3a). The number of such tuples is chosen so that the total number of tuples of each client is the same and equal to some constant number.

First, it is clear that if the client is honest, the server will accept the tolling cost.

Theorem 1 *If the server responds with "ACCEPT", the protocol in Figure 3 results in the correct tolling cost and respects the driver's location privacy.*

Proof: Assume that the client has provided an incorrect tolling cost in step 3b. Note first that all decommitment keys provided to the server must be correct; otherwise the server would have detected this when checking that the commitment was computed correctly. Then, at least one of the following data provided by the client provides has to be incorrect:

- The encryption of the pairs (s_j, t_j) obtained from the server. For instance, the car could have removed some entries with high cost so that the server computes a lower total cost in step iv).
- The computation of the total toll $COST$. That is, $COST \neq \sum_{v_i=s_j} t_j$. For example, the car may have reported a smaller cost.

For if both are correct, the tolling cost computed must be correct.

During each round, the server chooses to test one of these two conditions with a probability of $1/2$. Thus, if the tolling cost is incorrect, the server will detect the misbehavior with a probability of at least $1/2$. As discussed, the detection probability increases exponentially in the number of rounds.

For location privacy, we prove that the server gains no significant additional information about the car's data other than the tolling cost and the number of tuples involved in the cost (and see above for how to avoid the latter). Let us examine the information the server receives from the client:

Step (1c): The commitments $c(k)$ and $c(f_k(v_i))$ do not reveal information by the definition of a commitment scheme.

Step (i): $c(t_j)$ does not reveal information by the definition of a commitment scheme. By the definition of the pseudorandom function, $f_k(s_i)$ looks random.

Step (iii): If $b = 0$, the client will reveal k and t_j and no further information from the client will be sent to the server in this round. However, the values of $f_k(v_i)$ remain committed so the server has no other information about v_i other than these committed values, which do not leak information. If $b = 1$, the client reveals $f_k(v_i)$. However, since k is not revealed, the server does not know which pseudorandom function was used and due to the pseudorandom function property, these numbers look random. Providing D only provides decommitment to the sum of the tolls which is the result of the function, and no additional information is leaked (ie. in the case of the Pedersen scheme).

Information across rounds: A different pseudorandom function is used during every round so the information from one round cannot be used in the next round. Furthermore, the commitment to the same value in different rounds will be different and look random.

Therefore, we support our definition of location privacy because the road pricing protocol does not leak any additional information about whom the tuple tags belong to and the cars generated the tags randomly; therefore, our database is indistinguishable from a database without tags. □

The protocol is linear in the number of tuples the car commits to during registration and the number of tuples received from the server in step 3a. It is easy to modify slightly the protocol to reduce the number of tuples that need to be downloaded as discussed in Section 6.

Point tolls (replacement of tollbooths). The predominant existing method of assessing road tolls comes from point-tolling; in such schemes, tolls are assessed at particular points, or linked to entrance/ exit pairs. The latter is commonly used to charge for distance traveled on public highways. Such tolling schemes are easily handled by our protocol; tuples are generated corresponding to the tolling points. Tolls that depend on the entrance/ exit pairs can be handled by uploading a pair of tuples with the same ID; we discuss this refinement in detail for computation of speed below in Section 5.2. The tolling points can be "virtual", or alternatively an implementation can utilize the existing E-Zpass infrastructure:

- The transponder knows a list of places where tuples need to be generated, or simply generates a tuple per intersection using GPS information.
- An (existing) roadside router infrastructure at tolling places can signal cars when to generate tuples.

Other tolls. Another useful toll function is charging cars for driving in certain regions. For example, cars can be charged for driving in the lower Manhattan core, which is frequently congested. One can modify the tolling cost protocol such that the server assigns a cost of 1 to every tuple inside the perimeter of this region. If the result of the function is positive, it means that the client was in the specific region.

5.2 Speeding tickets

In this application, we wish to detect and charge a driver who travels above some fixed speed limit L. For simplicity, we will initially assume that the speed limit is the same for all roads, but this is not required by our protocol and we will discuss at the end how to relax this constraint. The idea is to cast speed detection as a tolling problem, as follows.

We modify the driving phase to require that the car uses each random vehicle tag v_i twice in succession; thus the car will upload pairs of linked path tuples. The server can compute the speed from a pair of linked tuples, and so during the reconciliation phase, the server assigns a cost t_i to each linked pair: if the speed computed from the pair is $> L$, the cost is non-zero, and it is zero otherwise. Now the reconciliation phase proceeds as discussed above. The spot check challenge during the reconciliation phase now requires verification that a consistent pair of tuples was generated, but is otherwise the same. If it deemed useful that the car reveal information about *where* the speeding violation occurred, the server can set the cost t_i for a violating pair to be a unique identifier for that speeding incident.

Since the number of linked tuples is half the total size of the database, the computational costs of this protocol are analogous to the costs of the tolling protocol and so the experimental analysis of that protocol applies in this case as well. There is a potential concern about additional side channels in the server's database associated with the use of linked tuples. Although the driver has the same guarantees as in the tolling application that her participation in the protocol does not reveal any information beyond the value of the function, the server has additional raw information in the form of the linkage. The positional information leaked in the linked tuple model is roughly the same as in the tolling model with twice the time interval between successive path tuples. Varying speed limits on different roads can be accommodated by having the prices t_i incorporate location.

5.3 Insurance premium computation

In this application, we wish to assign a "safety score" to a driver based on some function of their path which assesses their accident risk for purposes of setting insurance premiums. For example, the safety score might reflect the fraction of total driving time that is spent driving above 45 MPH at night. Or the safety score might be a count of incidents of rapid deceleration.

As in the speeding ticket example, it is straightforward to compute these sorts of quantities from the variant of the protocol in which we require repeated use of a vehicle identifier v_i on successive tuples.

If only a function of speed and position is required, the exact framework of the speeding ticket example will suffice. In order to accommodate detection of acceleration, we need to adjust the driving phase of the protocol again to now require that each tag be used on k successive tuples; local acceleration can be estimated from such a sequence. There is some possibility that acceleration events will fall across sequence boundaries and be missed using this methodology. If accuracy is at a premium, at the cost of a small increase in total database size we can require that the car upload overlapping tuples around the boundaries, where a given time-location pair would be uploaded with a pair of different tags corresponding to the different overlapping sequences.

6 Implementation

We implemented the road pricing protocol in C++ (577 lines on the server side and 582 on the client side). It consists of two modules, the client and the server. The source code is available at `http://cartel.csail.mit.edu/#vpriv`. We implemented the tolling protocol from Figure 3, where we used the Pedersen commitment scheme [21] and the random function family in [22], and a typical security parameter (key size) of 128 bits (for more security, one could use a larger key size although considering the large number of commitments produced by the client, breaking a significant fraction of them is very unlikely). The implementation runs the registration and reconciliation phases one after the other for one client and the server. Note that the protocol for each client is independent of the one for any other client so a logical server (which can be formed of multi-core or multiple commodity machines) could run the protocol for multiple clients in parallel.

In the protocols described above, the client downloads the entire set of tags (along with their associated costs) from the server. When there are many clients and correspondingly the set of tags is large, depending on the implementation this might impose unreasonable costs in terms of bandwidth and running time. In this section we discuss variants of the protocol in which these costs are reduced, at some loss of privacy.

Specifically, making a client's tags unknown among the tags of all users may not be necessary. For example, one might decide that a client's privacy would still be adequately protected if her tags cannot be distinguished in a collection of one thousand other clients' tags. Using this observation, we can trade off privacy for improved performance.

In the revised protocol, the client downloads only a subset of the total list of tags which satisfy some broad constraint. There are a variety of ways to constrain the set of tags to be downloaded, which we discuss below; here, we sketch the modifications to the protocol that are necessary presuming that the download policy has been set. For correctness, the client needs to prove that all of her tags are among the ones downloaded. This check can be performed similarly to the tolling protocol: the client encrypts the tags downloaded from the server with some of the random functions used in registration and reveals the corresponding car tag encryptions from registration, thus allowing the server to check inclusion of all these encrypted car tags in the encrypted subset of tags requested.

There are many ways in which the client could specify the subset of tags to download from the server. For instance, one way is to ask the server for some ranges of tags. For example, if the field of tags is between 0 and $(2^{128}-1)/2^{128}$, and the client has a tag of value around 0.5673, she can ask for all the tuples with tags in the range $[0.5672, 0.5674]$. The client can ask for an interval for each of her tags as well as for some junk tags. The client's tag should be in a random position in the requested interval. Provided that the car tags are random, in an interval of length ΔI, if there are $total$ tags, there will be about $\Delta I \cdot total$ tags. Alternatively, during registration clients could be assigned random "subset IDs" which are then subsequently used to download clusters of tags; the number of clients per subset ID can be adjusted to achieve the desired efficiency/ privacy characteristics.

7 Enforcement

The cryptographic protocol described in Section 5 ensures that a driver cannot lie about the result of the function to be computed given some private inputs to the function (the path tuples). However, when implementing such a protocol in a real setting, we need to ensure that the inputs to the function are correct. For example, the driver can turn off the transponder device on a toll road. The server will have no path tuples from that car on this road. The driver can then successfully participate in the protocol and compute the tolling cost only for the roads where the transponder was on and prove to the server that the cost was "correct".

In this section, we present a general enforcement scheme that deals with security problems of this nature. The enforcement scheme applies to any function computed over a car's path data. The idea is that the enforcement scheme verifies the correctness of the inputs to the protocol, namely the path.

The enforcement scheme needs to be able to detect a variety of driver misbehaviors such as using tags other than the ones committed to during registration, sending incorrect path tuples by modifying the time and location fields, failing to send path tuples, etc. To this end, we employ an end-to-end approach using sporadic *random spot checks*. We assume that at random places on the road, unknown to the drivers, there will be physical observations of a path tuple $\langle$`license plate,time,location`$\rangle$.

The essential point is that the spot check tuples are connected to the car's physical identifier, the license plate. For instance, such a spot check could be carried out by roving police cars that secretly record this information (perhaps similar to today's "speed traps").

The data from the spot check is then used to validate the entries in the server database. The reconciliation phase of the protocol from Section 5 is augmented with an additional challenge in which the driver is required to prove that she generated a path tuple that is sufficiently close to one observed during the spot check (and verify that the tag used in this tuple was one of the tags committed to during registration). Precisely, given a spot check tuples (t_c, ℓ_c), the driver must prove she generated a tuple (t, ℓ) such that $|t - t_c| < \Omega_1$ and $|\ell - \ell_c| < (\Omega_2)|t - t_c|$, where Ω_1 is a threshold related to the tuple production frequency and Ω_2 is a threshold related to the maximum rate of travel.

This proof can be performed in zero knowledge, although since the spot check reveals the car's location at that point, this is not necessary. The driver can just present as a proof the tuple it uploaded at that location. If the driver did not upload such a tuple at the server around the observation time and place, she will not be able to forge one due to the commitment during the registration phase. The server may allow a threshold number of tuples to be missing in the database to make up for accidental errors.

Intuitively, we consider that the risk of being caught tampering with the protocol is akin to the current risk of being caught driving without a license plate or speeding. It is also from this perspective that we regard the privacy violation associated with the spot check method: the augmented protocol by construction reveals the location of the car at the spot check points. However, as we will show in Section 8, the number of spot checks needed to detect misbehaving drivers with high probability is very small. This means that the privacy violation is limited, and the burden on the server (or rather, whoever runs the server) of doing the spot checks is manageable.

The spot check enforcement is feasible for organizations that can afford widespread deployment of such spot checks; in practice, this would be restricted principally to governmental entities. For some applications such as insurance protocols, this assumption is unrealistic (although depending on the nature of insurance regulation in the region in question it may be the case that insurance companies could benefit from governmental infrastructure).

In this case, the protocol can be enforced by requiring auditable tamper-evident transponders. The transponder should run correctly the registration and driving phase. Correctness during the reconciliation phase is ensured by the cryptographic protocol. The insurance company can periodically check if the transponder has been tampered with (and penalize the driver if necessary). To handle the fact that the

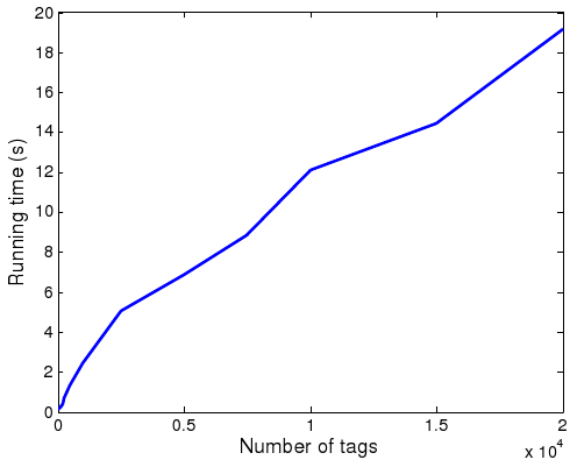

Figure 4: The running time of the road pricing protocol as a function of the number of tags generated during registration for one round.

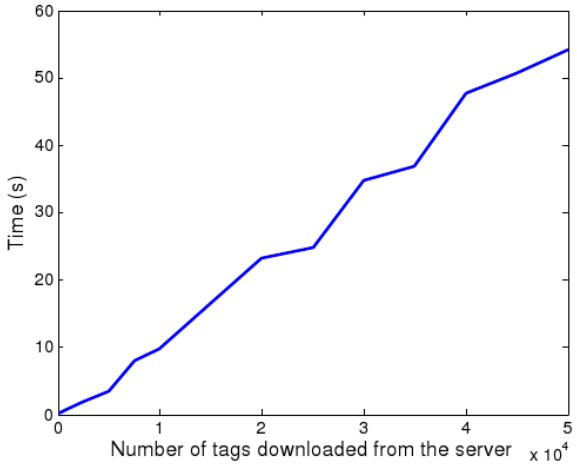

Figure 5: The running time of the road pricing protocol as a function of the number of tuples downloaded from the server during the reconciliation phase for one round.

driver can temporarily disable or remove the transponder, the insurance company can check the mileage recorded by the transponder against that of the odometer, for example during annual state inspections.

8 Evaluation

In this section we evaluate the protocols proposed. We first evaluate the implementation of the road pricing protocol. We then analyze the effectiveness of the enforcement scheme using theoretical analysis in Section 8.4.1 and with real data traces in Section 8.4.2.

8.1 Execution time

We evaluated the C++ implementation by varying the number of random vehicle tags, the total number of tags seen at the server, and the number of rounds. In a real setting, these numbers will depend on the duration of the reconciliation period and the desired probability of detecting a misbehaving client. We pick random tags seen by the server and associate random costs with them. In our experiments, the server and the clients are located on the same computer, so network delays are not considered or evaluated. We believe that the network delay should not be an overhead because we can see that there are about two rounds trips per round. Also, the number of tuples downloaded by a client from the server should be reasonable because the client only downloads a subset of these tuples as discussed in Section 6. We are concerned primarily with measuring the cryptographic overhead.

8.2 Execution time

Figures 4, 5, and 6 show the performance results on a dual-core processor with 2.0 GHz and 1 GByte of RAM. Memory usage was rarely above 1%. The execution time for a challenge bit of 0 was typically twice as long as the one for a challenge type of 1. The running time reported is the total of the registration and reconciliation times for the server and client, averaged over multiple runs. The graphs show an approximately linear dependency of the execution time on the parameters chosen. This result makes sense because all the steps of the protocol have linear complexity in these parameters.

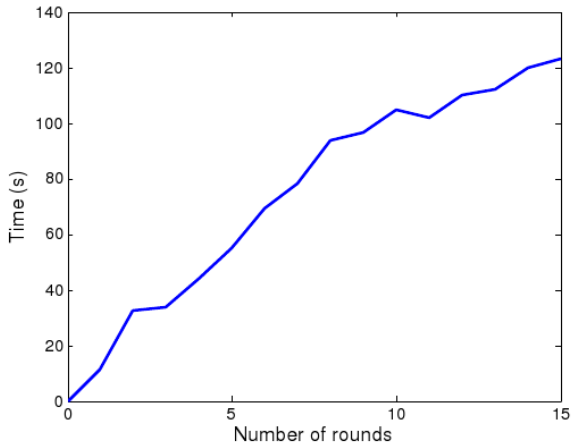

Figure 6: The running time of the road pricing protocol as a function of the number of rounds used in the protocol. The number of tags the car uses is 2000 and the number of tuples downloaded from the server is 10000.

In our experiments, we generated a random tag on average once every minute, using that tag for all the tuples collected during that minute. We choose that time because the average speed of a car traveling in urban and suburban areas in the US is about 25 MPH (according to the DoT). The average number of miles per car per year in the US is 12,000 miles, which means that each month sees about 40 hours of driving per car. Picking a new tag once per minute leads to $40 \times 60 = 2400$ tags per car per month (one month is the reconciliation period that makes sense for our applications).

We propose that a car downloads $10,000$ tuples from the server for the tolling protocol (note that these are only tuples with non-zero tolling cost). Assume a person roughly drives through 50 toll roads per month. Assuming no side channels, the probability of guessing which tuples belong to a car in this setting is $1/\binom{10000}{50}$, which is very small. Even if some of the traffic patterns of some drivers are known, the 50 tuples of the driver would be mixed in other 10000.

If the protocol uses 10 rounds (corresponding to a detection probability of 99.9%), the running time will be about $10 \cdot 10 = 100$ seconds, according to Figure 5. This is a very reasonable latency for a task that is done once per month and it is orders of magnitude less than the latency of the generic protocol [2] evaluated below. The server's work is typically less than half of the aggregate work, that is, 50 seconds. Downloading $10,000$ tuples (each about 50 bytes) at a rate of $10Mb/s$ yields an additional delay of 4 seconds. Therefore, one similar core could handle 30 days per month $\cdot 86400$ seconds per day / 54 seconds per car = 51840 cars per month. For 1 million cars, one needs $10^6/51840 \approx 21$ similar cores; this computation suggests our protocol is feasible for real deployment.

8.3 Comparison to Fairplay

Fairplay [29] is a general-purpose compiler for producing secure two-party protocols that implement arbitrary functions (specified in a subset of C). It generates circuits using Yao's classic work on secure two-party computation [24]. We implemented a simplified version of tolling protocol in Fairplay. Each party has a set of tuples and they compute the intersection of these tuples. The tolling cost is the number of tuples in the intersection (we made this simplification because the Fairplay protocol was prohibitively slow with a more complicated protocol). We found that the performance and resource consumption of Fairplay were untenable for very small-sized instances of this problem. The Fairplay program ran out of 1 GB of heap space for a server database of only 20 tags, and compiling and running the protocol in such a case required over 20 minutes. In comparison, our protocol runs with about 10K downloaded from the server tuples in 100s, which yields a difference in performance of *four orders of magnitude*. In addition, the oblivious circuit generated in this case was over 5 MB, and the scaling (both for memory and latency) appeared to be worse than linear in the number of tuples. There have been various refinements to aspects of Fairplay since its introduction,

but they do not substantially change the general conclusion that the general-purpose implementation of the relevant protocol is many orders of magnitude slower than VPriv.

8.4 Enforcement effectiveness

We now analyze the effectiveness of the enforcement scheme both analytically and using trace-driven experiments. We would like to show that the time a motorist can drive ilegally and the number of required spot checks are small. We will see that the probability to detect a misbehaving driver grows exponentially in the number of spot checks, making the number of spot checks logarithmic in the desired detection probability. This result is attractive from the dual perspectives of implementation cost and privacy preservation.

8.4.1 Analytical evaluation

We perform a probabilistic analysis of the time a motorist can drive illegally as well as the number of spot checks required. Let p be the probability that a driver undergoes a spot check in a one-minute interval. Let m be the number of minutes until a driver is detected with a desired probability. The number of spot checks a driver undergoes is a binomial random variable with parameters (p, m), pm being its expected value.

The probability that a misbehaving driver undergoes at least one spot check in m minutes is

$$\Pr[\text{spot check}] = 1 - (1-p)^m. \tag{1}$$

Figure 7, shows the number of minutes a misbehaving driver will be able to drive before it will be observed with high probability. This time decreases exponentially in the probability of a spot check in each minute. Take the example of $p = 1/100$. In this case, each car has an expected time of 100 minutes of driving until it undergoes a spot check and will be observed with 95% probability after about 8 hours of driving, which means that overwhelmingly likely the driver will not be able to complete a driving period of a month without being detected.

However, a practical application does not need to ensure that the cars upload tuples on all the roads. In the road pricing example, it is only necessary to ensure that cars upload tuples on toll roads. Since the number of toll points is usually only a fraction of all the roads, a relatively small number of spot checks will increase the probability of detection considerably. For example, if we have a spot check at one tenth of the tolling roads, after 29 minutes, each driver will undergo a spot check with 95% probability.

Furthermore, if the penalty for failing the spot check test is high, a small number of spot checks would suffice because even a small probability of detecting each driver would eliminate the incentive to cheat for many drivers. In order to ensure compliance by rational agents, we simply need to ensure that the penalty associated with noncompliance, β, is such that $\beta(\Pr[\text{penalization}]) > \alpha$, where α is the total toll that could possibly be accumulated over the time period. Of course, evidence with randomized law enforcement suggests strongly that independent of β, $\Pr[\text{penalization}]$ needs to be appreciable (that is, a driver must have confidence that they *will* be caught if they persist in flouting the compliance requirements).

If some tuples are lost in transit from client to server, the client can be given the choice of checking if all her tuples are included in the server's database and, if not, to make amendments before the protocol for the desired function begins. Only after the client has confirmed the tuples, will the information gathered from the spot check be verified for consistency with the server's database, after which the protocol for the desired function will proceed.

Nevertheless, even if we allow for a threshold t of tuples to be lost before penalizing a driver, the probability of detection is still exponential in the driving time as follows.

$$\Pr[\text{penalization}] \;=\; 1 - \sum_{i=0}^{t} \binom{m}{i} p^i (1-p)^{m-i} \geq 1 - e^{\frac{-(t-mp)^2}{2mp}},$$

where the last inequality uses Chernoff bounds.

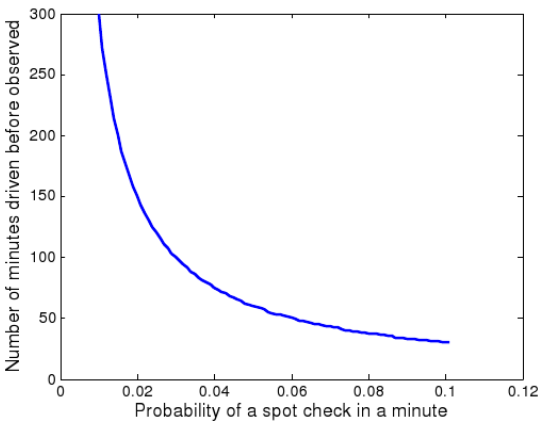

Figure 7: The time a motorist can drive illegally before it undergoes a spot check with a probability 95% for various values of p, the probability a driver undergoes a spot check in a minute.

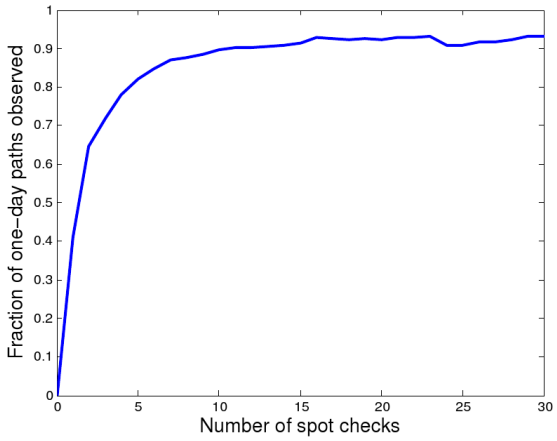

Figure 8: The fraction of one-day paths observed out of a total of 4826 one-day paths as a function of the total number of police cars placed.

8.4.2 Experimental evaluation

We now evaluate the effectiveness of the enforcement scheme using a trace-driven experimental evaluation. We obtained real traces from the CarTel project testbed [13], containing the paths of 27 limousine drivers mostly in the Boston area, though extending to other MA, NH, RI, and CT areas, during a one-year period (2008). Each car drives many hours every day. The cars carry GPS sensors that record location and time. We match the locations against the Navteq map database. The traces consist of tuples of the form (car tag, segment tag, time) generated at intervals with a mean of 20 seconds. Each segment represents a continuous piece of road between two intersections (one road usually consists of many segments).

We model each spot check as being performed by a police car standing by the side of a road segment. The idea is to place such police cars on certain road segments, to replay the traces, and verify how many cars would be spot-checked.

We do not claim that our data is representative of the driving patterns of most motorists. However, these are the best real data traces we could obtain with driver, time, and location information. We believe that such data is still informative; one might argue that a limousine's path is an aggregation of the paths of the different individuals that took the vehicles in one day.

It is important to place spot checks randomly to prevent misbehaving drivers from knowing the location of the spot checks and consequently to behave correctly only in that area. One solution is to examine traffic patterns and to determine the most frequently travelled roads. Then, spot checks would be placed with higher probability on popular roads and with lower probability on less popular roads. This scheme may not observe a malicious client driving through very sparsely travelled places; however, such clients may expend fuel and time resources by driving through these roads and they most likely do not even have to pay the toll. More sophisticated placement schemes are possible; here, we are primarily concerned with showing the ability to observe most traffic with remarkably few spot checks.

Consider the following experiment: we use the traces from a month as a training phase and the traces from the next month as a testing phase, for each month except for the last one. The first month is used to determine the first 1% (≈ 300) popular sites. We choose an increasing number of police cars to be placed

randomly at some of these sites. Then, in the testing phase we examine how many drivers are observed in the next month. We perform this experiment for an increasing number of police cars and for each experiment we average the results over fifty runs. In order to have a large sample, we consider the paths of a driver in two different days as the paths of two different drivers. This yields 4826 different one-day traces.

Figure 8 illustrates the data obtained. We can see that the fraction of paths observed increases very fast at the beginning; this is explained by the exponential behavior discussed in Section 8.4.1. After 10 spot checks have been placed, the fraction of paths observed grows much slower. This is because we are only considering 1% of the segments traveled by the limousine drivers. Some one-day paths may not be included at all in this set of paths. Overall, we can see that this algorithm requires a relatively small number of police cars, namely 20, to observe $\approx 90\%$ of the 4826 one-day paths.

Our data unfortunately does not reflect the paths of the entire population of a city and we could not find such extensive trace data. A natural question to ask would be how many police cars would be needed for a large city. We speculate that this number is larger than the number of drivers by a sublinear factor in the size of the population; according to the discussion in Section 8.4.1, the number of spot checks increases logarithmically in the probability of detection of each driver and thus the percentage of drivers observed.

9 Security analysis

In this section, we discuss the resistance of our protocol to the various attacks outlined in Section 3.2.

Client and intermediate router attacks. Provided that the client's tuples are successfully and honestly uploaded at the server, the analysis of Section 5 shows that the client cannot cheat about the result of the function. To ensure that the tuples arrive uncorrupted, the client should encrypt tuples with the public key of the server and anonymously sign them (e.g., via a group signature scheme). To deal with dropped tuples or tuples modified en route, the drivers should make sure that all their tuples are included in the subset of tuples downloaded from the server during the function computation. If some tuples are missing, the client can upload them to the server. These measures overcome any misbehavior on the part of intermediate routers.

The spot check method (backed with an appropriate penalty) is a strong disincentive for client misbehavior. An attractive feature of the spot check scheme is that it protects against attacks involving bad tuple uploads by drivers. For example, drivers cannot turn off their transponders because they will fail the spot check test; they will not be able to provide a consistent tuple. Similarly, drivers cannot use invalid tags (synthetic or copied from another driver), because the client will then not pass the spot checks; the driver did not commit to such tags during registration.

If two drivers agree to use the same tags (and commit to them in registration), they will both be responsible for the result of the function (i.e., they will pay the sum of the tolling amounts for both of them).

Server misbehavior. Provided that the server honestly carries out the protocol, the analysis of Section 5 shows that it cannot obtain any additional information from the cryptographic protocol. A concern could be that the server attempts to track the tuples a car sends by using network information (e.g., IP address). Well-studied solutions from the network privacy and anonymization literature can be used here, such as Tor [25], or onion routing [37].

Another issue is the presence of side channels in the anonymized tuple database. As discussed in Section 2, a number of papers have demonstrated that in low-density regions it is possible to reconstruct paths with some accuracy from anonymized traces [33, 34, 35]. As formalized in Definition 1, our goal in this paper was to present a protocol that avoids leaking any additional information beyond what can be deduced from the anonymized database. The obvious way to prevent this kind of attack is to restrict the protocol so

that tuples are uploaded (and spot checks are conducted) only in areas of high traffic density. An excellent framework for analyzing potential privacy violations has been developed in [31, 30], which use a *time to confusion* metric that measures how long it takes an identified vehicle to mix back into traffic. In [30], this is used to design traffic information upload protocols with exclusion areas and spacing constraints so as to reduce location privacy loss.

Recall that in Section 5, we assumed that the server is a passive adversary: it is trusted not to change the result of the function, although it tries to obtain private information. A malicious server might dishonestly provide tuples to the driver or compute the function f wrongly. With a few changes to the protocol, however, VPriv can be made resilient to such attacks.

- The function f is made public. In Figure 3, step 3a), the server computes the tolls associated to each tuple. A malicious server can attach any cost to each tuple, and to counteract this, we require that the tolling function is public. Thus, the client can compute the cost of each tuple in a verifiable way.
- For all the client commitments sent to the server, the client must also provide to the server a signed hash of the ciphertext. Then, the server cannot change the client's ciphertext without being observed.
- When the server sends the client the subset of tuples in Step 3a, the server needs to send a signed hash of these values as well. Then, the server cannot change his mind about the tuples provided.
- The server needs to prove to a separate entity that the client misbehaved during enforcement before penalizing it (eg. insurance companies must show the tamper-evident device).

Note that it is very unlikely that the server could drop or modify the tuples of a specific driver because the server does not know which ones belong to the driver and would need to drop or modify a large, detectable number of tuples. If the server rejects the challenge information of the client in Step iv) when it is correct, then the client can prove to another person that its response to the challenge is correct.

10 Conclusion

In this paper, we presented VPriv, a practical system to protect a driver's location privacy while efficiently supporting a range of location-based vehicular services. VPriv combined cryptographic protocols to protect the location privacy of the driver with a spot check enforcement method. A central focus of our work was to ensure that VPriv satisfies pragmatic goals: we wanted VPriv to be efficient enough to run on stock hardware, to be sufficiently flexible so as to support a variety of location-based applications, to be implementable with many different physical setups. and to resist a wide array of physical attacks. We verified through analytical results and simulation using real vehicular data that VPriv realized these goals.

Acknowledgments. We thank the members of the CarTel project, especially Jakob Eriksson, Sejoon Lim, and Sam Madden for the vehicle traces, and Seth Riney of PlanetTran. David Andersen, Sharon Goldberg, Ramki Gummadi, Sachin Katti, and Petros Maniatis provided many useful comments on this paper. We thank Robin Chase and Roy Russell for helpful discussions.

References

[1] P.F. Riley, *The Tolls of Privacy: An Underestimated Roadblock for Electronic Toll Collection Usage*, Proceedings of the Third International Conference on Legal, Security + Privacy Issues in IT, 2008.

[2] A.J. Blumberg and R. Chase, *Congestion Pricing That Respects "Driver Privacy"*, Proc. ITSC 2005.

[3] A.J. Blumberg, L.S. Keeler, and a. shelat, *Automated traffic enforcement which respects "driver privacy"*, Proc. ITSC 2004.

[4] J. Camenisch, S. Hohenberger, and A. Lysyanskaya. *Balancing Accountability and Privacy Using E-Cash*, SCN 2006.

[5] S. Rass, S. Fuchs, M. Schaffer, and K. Kyamakya, *How To Protect Privacy In Floating Car Data Systems*, Proceedings of the fifth ACM international workshop on VehiculAr Inter-NETworking, 2008.

[6] L. Sweeney, *k-anonymity: a model for protecting privacy*, International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems, v.10 n.5, 2002.

[7] C. Dwork, *Differential Privacy: A Survey of Results*, TAMC 2008: 1-19.
[8] D. Chaum, *Security without identification: transaction systems to make big brother obsolete*, CACM 28(10), 1985.
[9] A. Lysyanskaya, R.L. Rivest, A. Sahai, and S. Wolf, *Pseudonym systems*, Selected Areas in Cryptography (Howard M. Heys and Carlisle M. Adams, eds.), Lecture Notes in Computer Science, vol. 1758, Springer, 2000
[10] D. Goodin, *Microscope-wielding boffins crack tube smartcard*, `http://www.theregister.co.uk/2008/03/12/mifare_classic_smartcard_crack/`.
[11] O. Goldreich, S. Micali, and A. Wigderson, *Proofs That Yield Nothing but Their validity or All Languages in NP Have Zero-knowledge Proof Systems*, Journal of the ACM, Volume 38, Issue 3, p.690-728, July 1991.
[12] S. Goldwasser, S. Micali, and C. Rackoff. *The knowledge complexity of interactive proof-systems*, Proceedings of 17th Symposium on the Theory of Computation, Providence, Rhode Island. 1985.
[13] B. Hull, V. Bychkovsky, K. Chen, M. Goraczko, A. Miu, E. Shih, Y. Zhang, H. Balakrishnan, and S. Madden, *CarTel: A Distributed Mobile Sensor Computing System*, `http://cartel.csail.mit.edu`, in ACM SenSys, 2006.
[14] T. Litman, *London Congestion Pricing*, 2006.
[15] J. Miller, *With Cameras on Every Corner, Your Ticket Is in the Mail*, New York Times, January 6, 2005.
[16] R. Salladay, *DMV Chief Backs Tax by Mile*, Los Angeles Times, November 16, 2004.
[17] G. Zhong, I. Goldberg, and U. Hengartner, *Louis, Lester and Pierre: Three Protocols for Location Privacy*, 7th Privacy Enhancing Technologies Symposium, June 2007.
[18] E. Bangerter, J. Camenisch, and A. Lysyanskaya. *A Cryptographic Framework for the Controlled Release of Certified Data*, Security Protocols Workshop 2004.
[19] *E-ZPass*, `http://www.ezpass.com/index.html`.
[20] A. C. Yao, *Protocols for Secure Computations* (Extended Abstract), FOCS 1982: 160-164.
[21] T. P. Pedersen, *Non-Interactive and Information-Theoretic Secure Verifiable Secret Sharing*, Springer-Verlag, 1998.
[22] M. Naor and O. Reingold, *Number-Theoretic Constructions of Efficient Pseudo-Random Functions*, Journal of the ACM, Volume 51, Issue 2, p. 231-262, March 2004.
[23] Environmental Defense Fund, *Pay-As-You-Drive (PAYD) Auto Insurance*, `http://www.edf.org/article.cfm?ContentID=2205`.
[24] A. Yao, *Protocols for secure communications*. Proc. 23rd IEEE FOCS, 1982, p. 160-164.
[25] R. Dingledine, N. Mathewson, and P. Syverson. *Tor: The Second-Generation Onion Router*, USENIX Sec. Symp., USENIX Association 2004.
[26] U.S. Department of Transportation, National Transportation Statistics, `http://www.bts.gov/publications/national_transportation_statistics/`.
[27] G. Brassard, D. Chaum, and C. Crepeau, *Minimum Disclosure Proofs of Knowledge*, JCSS, 37, pp. 156-189, 1988.
[28] J. Eriksson, H. Balakrishnan, and S. Madden, *Cabernet: Vehicular Content Delivery Using WiFi*, MOBICOM, 2008.
[29] D. Malkhi, N. Nisan, B. Pinkas, and Y. Sella, *Fairplay - a secure two-party computation system*, USENIX Sec. Symp., USENIX Association 2004.
[30] B. Hoh, M. Gruteser, R. Herring, J. Ban, D. Work, J.-C. Herrera, A. Bayen, M. Annavaram, and Q. Jacobson. *Virtual Trip Lines for Distributed Privacy-Preserving Traffic Monitoring*, Mobisys, 2008.
[31] B. Hoh, M. Gruteser, H. Xiong, and A. Alrabady. *Preserving privacy in GPS traces via uncertainty-aware path cloaking*, ACM CCS 2007.
[32] M. Gruteser and D. Grunwald. *Anonymous usage of location-based services through spatial and temporal cloaking*, ACM MobiSys, 2003.
[33] B. Hoh, M. Gruteser, H. Xiong, and A. Alrabady. *Enhancing security and privacy in trafc-monitoring systems*, IEEE Pervasive Computing, 5(4):38-46, 2006.
[34] J. Krumm. *Inference attacks on location tracks*, Pervasive 2007.
[35] M. Gruteser and B. Hoh. *On the anonymity of periodic location samples*, Pervasive 2005.
[36] B. Gedik and L. Liu. *Location privacy in mobile systems: A personalized anonymization model*, 25th IEEE ICDCS 2005.
[37] D. Goldschlag, M. Reed, and P. Syverson. *Onion routing for anonymous and private internet connections,* CACM, 42(2), 1999.