VPriv: Protecting Privacy in Location-Based Vehicular Services

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

Andrew Blumberg
Department of Mathematics
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 locational 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 locational privacy of drivers. The starting point is the observation that in many applications, some centralized server needs to compute a function of a car's or user's* path*—a list of time-position tuples. VPriv provides two components: 1) a protocol to compute path functions 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 cars or users. Our implementation of VPriv is efficient enough to be run on inexpensive stock devices. Using analysis and simulation based on real vehicular data collected over two months from the CarTel project testbed of 27 taxis running in an urban area, we demonstrate that our protocol 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. Automated traffic enforcement (e.g., stop-light cameras) 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. "Pay-as-you-go" insurance programs in which insurance premiums are adjusted using information about driving behavior collected by GPS-equipped devices are being tested.

Unfortunately, along with the tremendous promise of these services comes very serious threats to the *locational privacy* of drivers (see Section 2 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 sizeable 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 uses *secure multi-party computations* to develop protocols for tolling and speed/delay estimation that do not compromise the locational privacy of the drivers. These cryptographic tools guarantee that a joint computation 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. We build on previous work which has proposed high-level protocols based on the use of such tools for preserving driver privacy [2], [3], [5]. Our main contribution here is the first implementation and experimental evaluation of multi-party secure protocols for specific functions computed over driving paths. Our approach is similar to the efficient protocol introduced in [17] for solving the problem of alerting an individual to nearby friends. More generally, this aspect of our work fits into the large body of efforts to devise practical multi-party secure protocols for particular problems.

The second component addresses a significant concern: making VPriv robust to attacks. Although we can prove security against "cryptographic attacks" as a consequence of the properties of the mathematical aspects 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. By identifying cars that incorrectly upload path information with high probability, the method ensures that *the argument* 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.

2. 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.

2.1. Framework

The participants in the system are *cars* and *a server*. For any given problem (tolling, traffic statistics estimation, insurance calculations, etc.), there is one logical server and many cars. The server wishes to compute some function f for any given car; f takes the path of the car generated during an *interaction interval* as its argument. The interaction interval is the time range over which the server wants to compute the function.

To compute f, the server must collect the set of points corresponding to the path traveled by the car over the desired interaction interval. Each point is a tuple with three fields:

$$\langle \texttt{tag, time, location} \rangle$$

Each car provides a sequence of such tuples to the server. The server computes f using the set of $\langle \texttt{time,location} \rangle$ pairs. If locational privacy were not a concern, the `tag` could uniquely identify the car. In our case, however, these tags should be chosen in a way that cannot be connected to an individual car.

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

1) *Usage-based tolls:* The server wishes to assess a path-dependent toll on the car. The toll is some function of the time and positions of the car, known to both the car 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.
2) *Automated speeding tickets:* The server wishes to detect 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 wishes to compute a "safety score" based on the car's path to determine insurance premiums. Specifically, the server wishes to compute some function of the time, positions, speed, and acceleration of the car. For example, we might wish to assess higher premiums on cars which persistently drive close to the speed limit, or have frequent rapid acceleration events.

We regard all of these applications as essentially similar examples of the basic problem of computing a localized cost function of the car's path represented as points. Here 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, the general reduction is not practical, and so by exploiting specifics of the problem we devise efficient protocols.

Our model is that each car has some way of obtaining the point tuples as it drives and delivering them to the server. Conceptually, imagine a node on the car gathering GPS information periodically (e.g., every second) and delivering it over some network. The delivery of this information need not be done in real-time and could happen sporadically, as long as it happens before the server computes the function. Moreover, the car may not actually carry a node; roadside devices could infer the presence of a car (e.g., cameras, RFID, etc.) and act as a proxy for this information. Thus, this abstract model 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.

2.2. Design goals

We have the following goals for the protocol between the car 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 $f(P_C)$.

Locational privacy. We want to ensure that the computation of f does not reveal the path of the car. We formalize our notion of locational privacy in this paper as follows:

Definition 1: **(Locational privacy)** *Let*

- $\mathbb{S}$ *denote the server's database consisting of tuples* $\langle$*tag, time, location*$\rangle$.
- $\mathbb{S}'$ *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.*
- $\mathcal{V}$ *denote the information sent by* C *to the server while executing the protocol, together with any other information owned or computed by the server during the computation of* f*(path of* C*).*

Then we say that the computation of f*(path of* C*) preserves the* locational privacy *of* C *if the server gains an insignificant amount of additional information about which tuples belong to* C *from* $\mathbb{S}$*,* $\mathcal{V}$ *and* f(*path of* C) *than from* $\mathbb{S}'$ *and* f(*path of* C).

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.

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

$$\langle \texttt{time, location} \rangle$$

without any identifying information. Note that this definition means that any covert 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. 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 even on inexpensive in-car devices. This goal can be hard to achieve; modern cryptographic protocols can be computationally intensive.

2.3. Threats

We assume that a car cannot trust the server with its locational privacy, the server does not trust the car or its user to compute any operation correctly, the server assumes that any device or software running on the car may have been tampered with, and there may be intermediate untrusted devices (e.g., roadside nodes, wireless access points, etc.) that lie between the car and the server. Both the car and server may have strong financial incentives to misbehave. The following (non-exhaustive) list mentions the kinds of attacks we are concerned with:

- The driver turns off or selectively disables the in-car node, so the car uploads no data or only a subset of the actual path data.
- The car or driver uploads synthetic data.
- The car eavesdrops on another car and attempts to masquerade as that car.
- Some intermediate router synthesizes false packets or systematically changes packets.

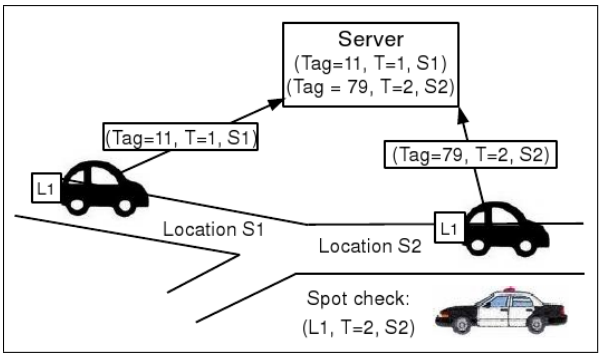

Figure 1: VPriv's system overview. A car with license plate L1 is traveling from Location S1 at time 1 to Location S2 at time 2 when it undergoes a spot check. It uploads path tuples to the server.

- The server inserts synthetic data claiming it has come from a particular car.

Thus, we are concerned not only with making sure that the protocol achieves our privacy and correctness goals when successfully carried out, but also with assuring that the car and server participate honestly in the protocol.

3. Architecture

This section gives an overview of the VPriv system and its components. As explained in the previous section, the abstract model consists of in-car nodes gathering periodic time-location tuples. To protect locational privacy, the server must not be able to associate the tags in the path tuples to a specific car or driver; otherwise the server could aggregate the tuples based on the tags and deduce the paths of individual cars. Thus, we require that the car chooses tags at random to prevent the server from knowing which points belong to the same path. However, these random tags will still be cryptographically bound to the car, as we will discuss. The car will not be able to disavow having produced a particular tuple and the server will not be able to match tags with specific cars.

VPriv consists of three key steps:

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 car's node (aka "car") 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 car then engages in a cryptographic *commitment scheme* with the server to commit the selected random tags. We describe the details of the commitment scheme we use in Section 3.1.
2) **Driving.** As the car drives, it gathers timestamped locations 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 4). In addition, VPriv relies on sporadic random spot checks to observe the physical locations of cars. This process generates tuples consisting of the actual license plate number, time, and location of observation. This information is cross-checked (during reconciliation) against the randomly tagged information sent by cars to determine misbehavior.
3) **Reconciliation.** This stage computes the function f, and happens at the end of each interaction interval. The server and the car engage in a secure two-party protocol in which:
   - They compute the desired function of the car's path, with the car providing its path as *private input*.
   - The car proves to the server in zero knowledge that its private input to the computation of the function is precisely the tuples that appeared in the database.

   Optionally, the server may challenge the driver to verify that the private input to the computation above is consistent with the spot check tuples.

This framework is analogous to the setup of [2], [3]. To implement this protocol, VPriv uses a set of modern cryptographic tools: secure two-party computations, a homomorphic commitment scheme, and random function families. We provide a brief overview of these tools below. The experienced reader may skip to Section 4, where we provide efficient realizations that exploit details of our restricted problem setting.

3.1. 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.

$COST$	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 $t_i > 0$.

Table 1: Notation.

- 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 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 them managed to learn the private information of another one other than what can be inferred from the result of the function. In the case of our 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 with no private data with respect for this protocol. The driver is malicious because she would like to lower the tolling cost; the server will behave correctly, though it may attempt to gain information about the driver's path. A **zero-knowledge proof** [12] is a related concept that involves proving the truth of a statement without revealing any other information.

A **random function family**[22] is a collection of functions $\{f_k\} : D \rightarrow R$ 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).
- f_k is indistinguishable from a function with random output for each input.

4. Protocols

This section presents a detailed description of the specific interactive protocol for our applications, making precise the preceding informal description. We present the interaction between the server and the client only within one round because all the other rounds are similar. 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.

4.1. Tolling protocol

We first introduce the notation in Table 1. For clarity, we present the protocol in a schematic manner in Figure 2. 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 obfuscated version of her tags on which the server can compute the tolling cost in ciphertext. The server has a way of verifying that the obfuscations provided by the client are 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 obfuscations are computed correctly, or compute the tolling cost on the vehicle tags using the obfuscations.

Note that this protocol also reveals the number of tolling tuples of the car because the server knows the size of the intersection. This can be seen from the number of $f_k(v_i) = f_k(s_j)$ in iv). The protocol can be amended to avoid this by having the motorist upload a random number of padding tuples along with each real tuple: these tuples have the same time and location, but a different tag. The server assigns a nonzero cost to only one of the tuples from that time and location, and the protocol otherwise proceeds as above.

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 2 results in the correct tolling cost and respects the driver's locational privacy.

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

- The obfuscation of the pairs $(s_i,\ t_i)$ obtained from the server. For instance, the car could have

Privacy-Preserving Computation of the Tolling Cost

1) **Registration phase:**
   a) For each car choose random vehicle tags, v_i, and a random function, f_k, by choosing k at random.
   b) Obfuscate the selected vehicle tags by computing $f_k(v_i), \forall i$, commits to the random function by computing $c(k)$, commits to the obfuscated 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 vehicle 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_i, for each random tags s_i observed in the last period based on the location and time where it was observed and sends (s_i, t_i) to the client only if $t_i > 0$.
   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. Obfuscate 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 all t_j to the server and proves that these are the values the client committed to in step (i) by providing $d(k)$ and $d(t_j)$. If $b = 1$, the client sends the obfuscations to 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 3.1, 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_i, t_i) have been correctly shuffled, obfuscated 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 2: VPriv's protocol for computing the tolling cost. The arrows indicate data flow.

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

Step (iii): If $b = 0$, the client will reveal k and t_i 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 random function was used and due to the random function property, these numbers look random. Providing D only provides decommitment to the sum of the tolls which is the result of the function. One needs to choose a homomorphic commitment or encryption with the property that D will not reveal any significant information about the any v_i.

Information across rounds: A different random 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 like random numbers.

Therefore, we support our definition of locational 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. We believe the latter is manageable in size because the server sends tuples whose costs are nonzero and the number of tolling roads is much smaller than the number of roads. Furthermore, the only tuples of interest are those from the last tolling period. If this number is still considered large, clients can exchange privacy for performance and constrain the download via statements such as "I have only been in Boston", subject to the enforcement protocol discussed in Section 5.

Notice that an algorithm that requires a time sublinear in the number of tuples at the server with nonzero cost has decreased privacy guarantees: this is because the server excludes from consideration some tuples, which means that the server knows or is being told that those tuples do not belong to the car.

4.2. 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 use 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 nonzero, 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 covert 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.

Clearly, varying speed limits on different roads can be accommodated by having the prices t_i incorporate location. A more subtle question is to try and detect distinct "instances" of speeding. To some degree, this can be handled by appropriate assignment of costs, but in general we leave the problem of formally specifying such a definition and constructing a protocol for it to future work.

In Section 4, we explained how to avoid leaking the number of tolling tuples by uploading a varying number of tuples instead of one at a time. The same solution can be applied here by uploading the same number of additional tuples for each of the two tuples

with the same tag.

4.3. 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.

5. Enforcement

The cryptographic protocol described in Section 4 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 is 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

$$\langle \texttt{license plate, time, location} \rangle$$

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 4 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). 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 begin 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 6, 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.

6. Evaluation

In this section we evaluate the protocols proposed. We implemented the road pricing protocol, which we

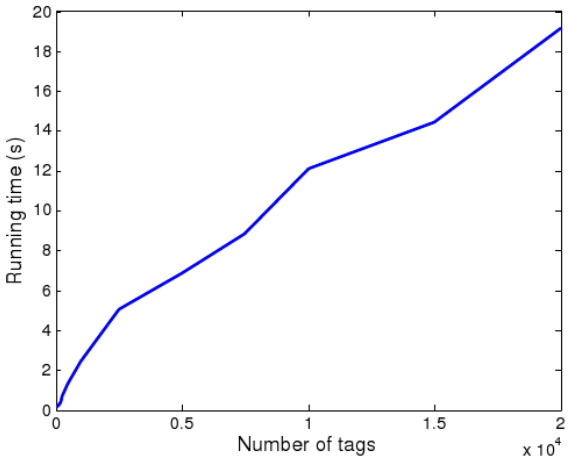

Figure 3: The running time of the road pricing protocol as a function of the number of tags generated during registration for one round.

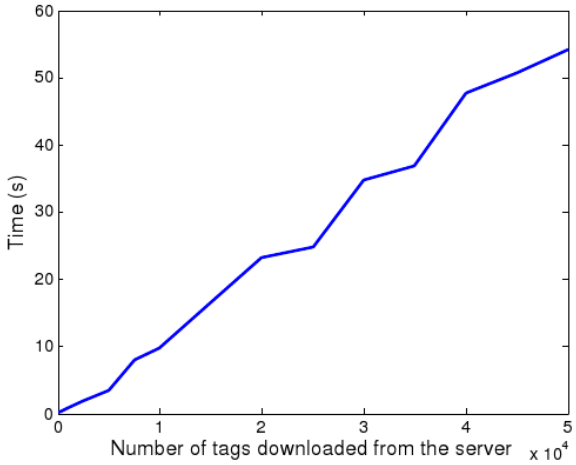

Figure 4: The running time of the road pricing protocol as a function of the number of tuples downloaded from the server during the reconciliation phase for one round.

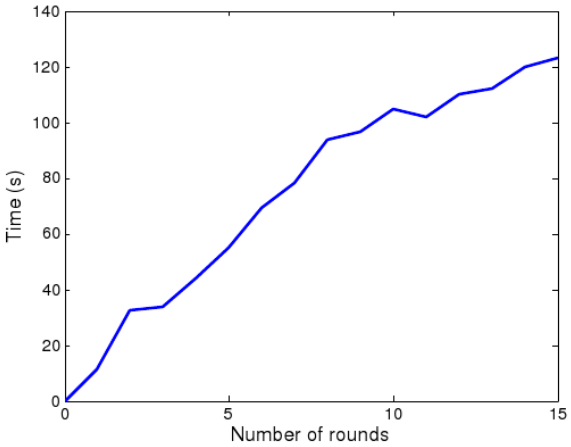

Figure 5: The running time of the road pricing protocol as a function of the number of rounds used in the protocol. The number of tags the car uses is 2000 and the number of tuples downloaded from the server is 10000.

evaluate in Section 6.1. We then analyze the effectiveness of the enforcement scheme using theoretical analysis in Section 6.3.1 as well as real data traces in Section 6.3.

6.1. Implementation

We implemented the road pricing protocol in C++. It consists of two modules, the car and the server, which interact according to the protocol described in Section 4.

We evaluated the protocol 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 small because the client only considers tuples with nonzero tolling cost from a certain region. We are concerned primarily with measuring the cryptographic overhead.

6.2. Execution time

We implemented the commitment scheme from scratch according to [21] and the random function according to [22]. We used a key size of 128 bits. Figures 3, 4, 5 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.

In our experiments, we generated a random tag on average once every minute, using that tag for all the tuples collected during that epoch. We choose that time because the average speed of a car traveling in urban and suburban areas in the US is about 25MPH. 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, 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 covert 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 4, which is an acceptable latency for a task done once per month.

6.3. 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 illegally 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.

6.3.1. 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

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

Figure 6, 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 will be observed with 95% probability after about 8 hours of driving, which means that

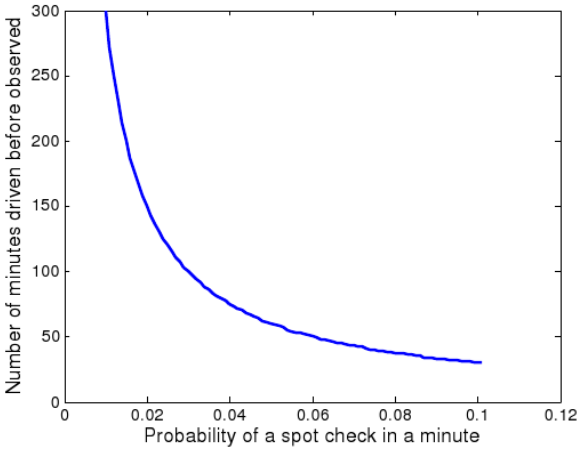

Figure 6: The time a motorist can drive illegally before it undergoes a spot check with a probability 95% for various values of p, the probability a driver undergoes a spot check in a minute.

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.

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

where the last inequality uses Chernoff bounds.

6.3.2. 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 two-month period (January and February 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 periods 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, assuming these motorists drive illegally.

We do not claim that our data is representative of the driving patterns of most motorists. However, this is the best real data traces we could obtain with driver, time, and location information. We believe that such data is still informative. One argument can be 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 non-deterministically 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 popular places. Then, spot checks would be placed with higher probability on popular roads and with lower probability on less popular roads.

Consider the following experiment: we use the traces from January 2008 as a training phase 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 February. 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 443 different one-day traces.

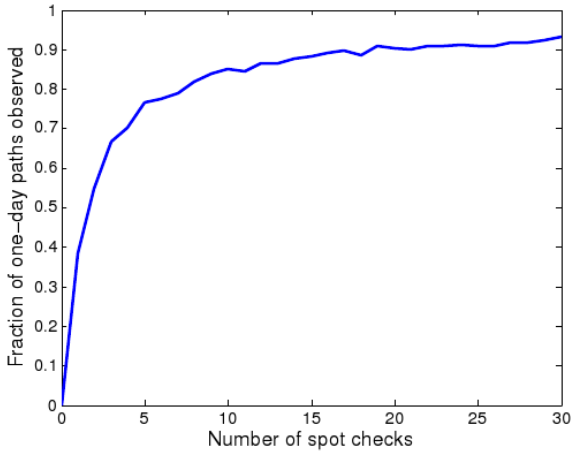

Figure 7: The fraction of one-day paths observed out of a total of 443 one-day paths as a function of the total number of police cars placed.

Figure 7 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 6.3.1. 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 443 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 6.3.1, the number of spot checks increases logarithmically in the probability of detection of each driver and thus the percentage of drivers observed.

7. Security analysis

In this section, we discuss the resistance of our protocol above to a selection of common attacks on the part of the driver, the intermediate routers (or monitor-

ing devices), and the server. As a first step, in order to prevent malicious intermediaries from modifying the tuples sent by a car to the server, we assume that the tuples are encrypted with the public key of the server.

In concert with the "spot check" methodology, this provides a remarkably powerful means of thwarting common attacks by the driver, particularly when paired with a legal penalty. One of the attractive features of this technique is that it is very general. For instance, it protects against cases when the driver does not upload data or uploads tuples with tags other than the ones obtained during registration.

The router can also claim it uploaded the tuples but instead maliciously delete them. If this is a concern, the server gives a chance to the driver to check which tuples were received and upload any missing ones before the payment protocol.

A more serious potential problem is that malicious routers in collusion with the server may attempt to track the tuples a car sends in the network in order to infer the path of a car. This is a particular concern when the driver wants to upload a large quantity of tuples at once. There is extensive literature on anonymizers that attempt to prevent this problem; for example Tor [25] is a known example.

A final issue is the presence of covert channels in the raw database, which the server can use to infer information about the clients. For example, the server can make obvious deductions upon receives a tuple from a street where only Bob lives. As discussed in Definition 1, our goal in this paper is simply to avoid leaking any additional information beyond what the server already knows.

8. Related work

Electronic tolling and public transit fare collection was one 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]. Unlike VPriv, the electronic cash approach is significantly less suitable for more fine-grained 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, aggregate traffic statistic collection.

There has been 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. However, these papers use multi-party computations and zero-knowledge proofs as a black box and refer to the literature for implemeting such protocols. Whereas the literature provides general guidelines on building such protocols, the resulting protocols are usually inefficient. Here, we exploit the specifics of locational privacy to design an efficient solution. On the other hand, in the social area of locational privacy there has been some recent work on efficient protocols for the problem of determining when friends are nearby [17]. A further issue with all of these protocols is the handling of noncompliance and physical attacks. The application constraints in [17] mostly obviate these concerns, and the traffic application papers either ignore the issue or impose stringent requirements on the monitoring infrastructure: notably, that the car's tuples are transmitted by trusted devices which can reliably know their own location, and are equipped with the ability to detect vehicles passing physically. We believe that our "spot check" methodology for comprehensively handling this problem is the proper solution, as it is efficient, flexible, and compatible with a wide range of implementation choices and problems.

Finally, there has been a good deal of work on the general question of maintaining the anonynimity and personal privacy of individuals who contribute to centralized databases, notably the work on k-anonynimity [6] and differential privacy [7]. In this paper we have not addressed these questions — our notion of locational privacy is independent of the issue of whether or not the mere collection of anonymized traffic data violates the privacy of drivers. We regard this as a fundamental avenue for future work.

9. Conclusion

In this paper, we presented VPriv, a practical system to protect a driver's locational privacy while efficiently supporting a range of location-based vehicular services. VPriv combined cryptographic protocols to protect the locational privacy of the driver with a spot check methodology to ensure compliance. 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, and to be implementable with many different physical setups. We verified through analytical results and simulation using real vehicular data that VPriv realized these goals. Furthermore, we showed that the spot check methodology makes VPriv resistant to a wide array of common physical attacks.

Acknowledgment

We thank the members of the CarTel project, especially Jakob Eriksson, Sejoon Lim, and Sam Madden for helping collect and manage the vehicular traces used in our evaluation, and Seth Riney of PlanetTran. We thank Robin Chase and Roy Russell for helpful discussions. This work was supported in part by the National Science Foundation under grants 0205445, 0716273, and 0520032.

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.* Security and Cryptography for Networks (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*, Communications of the ACM 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/doku.php, in Proc. 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 Symposium on Foundations of Computer Science (FOCS '82), 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*, Journal of Computer and System Sciences, vol. 37, pp. 156-189, 1988.