Provable and Practical Location Privacy for Vehicular and Mobile Systems

by

Raluca Ada Popa

B.S., Computer Science and Engineering (2009)
B.S., Mathematics (2009)
Massachusetts Institute of Technology

Submitted to the Department of Electrical Engineering and Computer Science
In partial fulfillment of the requirements for the degree of

Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

May 2010

Author . . . .
Department of Electrical Engineering and Computer Science
May 7, 2010

Certified by . . . .
Hari Balakrishnan
Professor of Electrical Engineering and Computer Science
Thesis Supervisor

Accepted by . . . .
Christopher J. Terman
Chairman, Department Committee on Graduate Students

Provable and Practical Location Privacy for Vehicular and Mobile Systems

by

Raluca Ada Popa

Submitted to the Department of Electrical Engineering and Computer Science
May 7, 2010
In partial fulfillment of the requirements for the degree of
Master of Engineering in Electrical Engineering and Computer Science

Abstract

In recent years, there has been a rapid evolution of location-based vehicular and mobile services (e.g., electronic tolling, congestion pricing, traffic statistics, insurance pricing, location-based social applications), which promise tremendous benefits to users. Unfortunately, most such systems pose a serious threat to the location privacy of users because they track each individual's path.

A question that arises naturally is how can we preserve location privacy of users while maintaining the benefits of such services? In this thesis, we address this question by tackling two general problems that are the foundation of many of the aforementioned services. The first problem is how to enable an untrusted server to compute agreed-upon functions on a specific user's path *without* learning the user's path. We address this problem in a system called *VPriv*. VPriv supports a variety of applications including electronic tolling, congestion pricing, insurance premium computation, and some kinds of social applications. The second problem is how to enable an untrusted server to compute aggregate statistics over all users' paths *without* learning any specific user's path. We tackle this problem in a system called *PrivStats*. With PrivStats, one can compute statistics such as traffic statistics (e.g., average speed at an intersection, average delay on a road, number of drivers at a location) or average ratings of a location in a social application. The computation and threat models for VPriv and PrivStats are different, and required markedly different solutions. For both systems, we provide formal definitions of location privacy and prove that our protocols achieve these definitions. We implemented and evaluated both systems, and concluded that they are practical on commodity hardware and smartphones.

Thesis Supervisor: Hari Balakrishnan
Title: Professor of Electrical Engineering and Computer Science

Pentru Tatiana, mama mea.
Mulţumesc.

Acknowledgments

This research was performed under the supervision of Professor Hari Balakrishnan and it is joint work with Professor Andrew J. Blumberg, from University of Texas, Austin. The work on VPriv was published at the 18th Usenix Security Symposium and the paper on PrivStats is in submission.

I would like to thank Hari for being an amazing advisor. I was greatly inspired by his excitement, vision, and creativity. Working with Hari was a great learning experience and a lot of fun. I am especially thankful for his constant and enthusiastic support.

My thesis work would not have been as rewarding without Andrew. I enjoyed a lot our persistent attempts to solve a problem and our weekly reading sessions on cryptography topics. Thanks a lot for your help and enthusiasm for the project!

I am deeply grateful to Barbara Liskov and Ron Rivest, who were my first research mentors. I wish to thank them for the many things they taught me, for their generosity in sharing their time with me, and for their support. They have greatly inspired me and became my role models.

I am profoundly indebted to Liviu and Cristina Iftode for their generous support and advice with moving for college to the US.

I would like to thank Alessandro Chiesa, my boyfriend, for his support, encouragement, and care. He is also a great friend and technical collaborator.

Thanks to my friend Doris Dobi for our enthusiastic discussions and lasting friendship.

I am grateful to Silvio Micali for having taught me cryptography and to Frans Kaashoek for his advice. The members of the CarTel project, especially Sam Madden, Jakob Eriksson, Sejoon Lim, and Arvind Thiagarajan provided access to data traces and useful feedback. In addition, I thank Seth Riney of PlanetTran for his partnership with the CarTel project, which enabled the collection of data used to evaluate the protocols in my thesis. David Andersen, Sharon Goldberg, Ramki Gummadi, Sachin Katti, Petros Maniatis, and the members of the PMG group at MIT have provided many useful comments. Also thanks to Yuanyuan Zhou and Nickolai Zeldovich.

This work was supported in part by the National Science Foundation under grants CNS-0716273 and CNS-0520032.

Overview

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The price tag for technology should not include a loss of privacy.
- James Dempsey,
Scottish politician.

In recent years, there has been a rapid growth of location-based vehicular and mobile services. These applications promise to improve driver experience, raise revenue for road transportation, reduce traffic congestion, as well as provide new opportunities for social networking. In particular, over the next few years, location-based vehicular services using a combination of in-car devices and roadside surveillance systems will likely 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 [32] (to reduce bottlenecks at toll plazas), congestion pricing [43](to generate revenue for transit improvements), automated traffic law enforcement, traffic statistics collection [38, 47, 42, 48] (for traffic prediction and optimization), "pay-as-you-go" insurance [39] (to adjust premiums based on driver behavior), and so on. Moreover, location-based social applications (of which there are currently more than a hundred [59]) enable users to select restaurants and stores that have good reviews [20] or play games in which users collect points [23].

Unfortunately, the tremendous promise of these services comes with serious threats to the *location privacy* of the participants. Some current implementations of these services involve pervasive tracking: clients upload their identifier together with location data. For instance, in the E-ZPass system [32], drivers upload their account ID every time they pass by a toll booth. As another example, in smartphone social applications, clients send their identifier or username to a centralized server with location updates. As a consequence, a centralized server belonging to these services can put together into a path all the location-time pairs received from each user, thus violating privacy. Any service that maintains all this information can easily be subject to abuse by people inside the organization running the service, attackers who break in the system, or by government edicts or legal subpoenas [1]. Such privacy concerns also reduce the willingness of individuals to adopt such systems (e.g., [58]).

In this thesis, we attempt to provide protocols that provably protect user location privacy while maintaining the practical functionality and benefits of such services. The contributions

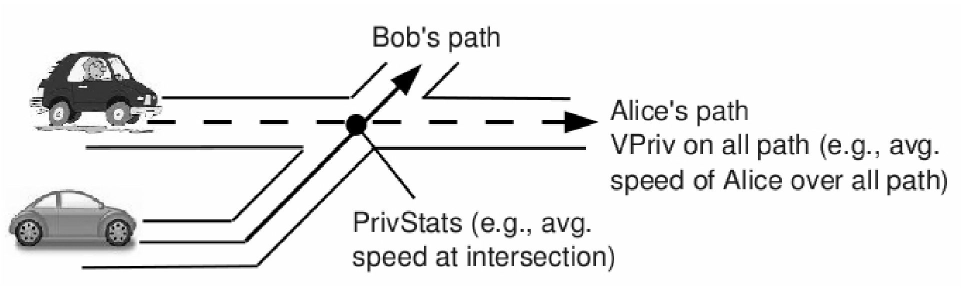

Bob's path
Alice's path
VPriv on all path (e.g., avg. speed of Alice over all path)
PrivStats (e.g., avg. speed at intersection)

Figure 1-1: Overall diagram of the roles of VPriv and PrivStats exemplified using the average speed case.

of this thesis are two systems, **VPriv** and **PrivStats**; they address two general problems that form the basis of most applications mentioned:

- VPriv enables an untrusted server to compute an agreed-upon function on a user's path without being able to learn the user's private path. Applications such as electronic tolling, congestion pricing, pay-as-you-go insurance, law enforcement, and some kinds of social networking games fall in this category.
- PrivStats enables an untrusted server to compute aggregate statistics over the paths of all the users, without being able to learn individual statistics or paths. Such statistics can be average speed at an intersection, average delays on certain roads, the number of drivers passing through an intersection (for congestion estimation purposes), or the average rating in a social network of a place.

Each of the two systems provides novel contributions as compared to related work. We overview these contributions in the following two subsections. Figure 1-1 provides an example of the usage of VPriv and PrivStats.

The goals for both VPriv and PrivStats seem paradoxical: how can one perform computation on data he does not know? The idea behind both systems is to combine the power of modern cryptographic notions such as zero-knowledge protocols (explained in Chapter 3 along with other tools we used) with engineering effort to construct new protocols and systems that are secure and practical.

We first designed VPriv to solve the problem of function computation on private paths. While working on VPriv, we realized that the problem of private statistics computation was also very important for many applications and was not properly addressed by VPriv or related work; VPriv was not applicable to the threat model and goals of a statistics computation application (as discussed in Section 2.3), so we required markedly different protocols. Moreover, we realized that neither our model nor related work addressed an important avenue for privacy leakage, side information, which we will describe below. This is what led to our second research project, PrivStats. Besides addressing a different problem than VPriv, PrivStats also enhances VPriv by providing stronger privacy definitions and protocols that resist to general side information leaks.

VPriv and PrivStats can support a wide range of applications. At a high level, they rely on a reasonably unrestrictive model: mobile nodes (smartphones, in-car devices, etc.)

equipped with GPS or other position sensors (and some reasonable computing and storage resources) periodically upload timestamped location coordinates and can communicate with a server. We present the precise model in Chapter 4. As such, VPriv and PrivStats can be applied to any applications that support this model. As discussed, a wide range of location-based vehicular applications falls within this model; this class of applications seems to be the most relevant for our systems and thus constitutes our running example. We designed VPriv with mostly the vehicular applications in mind, whereas we designed PrivStats to be more general. Nevertheless, certain social applications and some mobile systems can benefit from both systems as we explain in Chapter 7.

Code and more information about these projects can be found on the project's website, `http://nms.csail.mit.edu/projects/privacy/`.

1.1 VPriv Overview

VPriv is a practical system to protect a user's location privacy while efficiently supporting a range of location-based vehicular services. VPriv supports applications that compute functions over the *paths* traveled by individual clients. 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, as well as 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 computing sum of costs functions (tolling, speed or delay estimation) that protects the location privacy of the clients.* This protocol, which belongs to the general family of secure multi-party computations, guarantees that a joint computation between a server and a client can proceed correctly without revealing the private data of the parties involved. The result is that each driver is guaranteed that no other information about his 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 [7, 8, 57]; 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 three orders of magnitude slower than our implementation in our experiments (see Section 5.5.2), which makes them impractical.

Our main contribution here is the first *practically efficient* design, software implementation, and experimental evaluation of secure multi-party 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 [53] to allow the server to compute such sums using encrypted data.

The second component of VPriv addresses a significant concern: *making VPriv robust to physical 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 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.

1.2 PrivStats Overview

PrivStats aims to preserve privacy while allowing an *aggregator* to compute aggregate statistics over the paths of clients. An aggregate statistics is any function that can be computed on all the values received from participating clients. Our motivating examples are traffic applications which collect GPS position and/or speed samples along vehicle trajectories to determine current traffic delays (average speed, average delay) on road segments. Between nascent government efforts (e.g., the DOT "Intellidrive" initiative [51]), research efforts (e.g., CarTel [38], MobileMillenium [47], CommuteAtlanta [42]), and commercial systems, there is now a diverse array of systems that aim to provide such services.

A first attempt at a solution is to anonymize the time-location uploads by clients by removing any client identifier, leaving only the time and the location information. However, this solution has a number of serious deficiencies, including: *location privacy violations via inference*, *lack of resilience to side information*, and *lack of accountability.* It is possible to recover a surprising amount of information about the individuals paths of specific mobile clients from such time-location information [40, 34, 55] using simple inference algorithms. Side information (any out-of-bound information available at the server about a client), denoted SI, is problematic in many scenarios; as a simple example, if the aggregator knows that Alice is the only person living on a street, then speed updates on that street can easily be linked to Alice. In the presence of anonymous uploads, clients can now upload a high volume of fake data to bias the statistics in a desired direction.

In fact, any solution that has strong anonymity guarantees would seem to have even worse problems with accountability. Nonetheless, it turns out to be possible to balance accountability and location privacy. We designed, implemented, and evaluated PrivStats, the first system for computing aggregate statistics in the context of mobile, location-based applications that simultaneously achieves quantifiable accountability and provable protection of location privacy. In addition, PrivStats is specifically designed to work in settings where no compliance assumptions about the participants can be made.

PrivStats can be used to provide either *plausible deniability*, using our plausible deniability protocol (denoted PD), or *strict location privacy*, using our strict location privacy protocol (denoted SLP). PD does not have the strong security properties of SLP, but it is a very simple protocol; SLP provides strong privacy guarantees, but it has higher (yet reasonable) overhead.

Both protocols support a wide range of statistics. PD supports virtually any statistics that can be computed on the values uploaded by clients, while SLP supports most statistics needed in practice including count, average, sum, product, standard deviation, and conditional aggregations of all of these. Section 6.3 characterizes the statistics supported.

PrivStats makes two central contributions:

1. *Aggregate statistics protocols allowing provable guarantees of location privacy:* Unlike previous efforts, we treat the presence of side information as a first-order problem, and provide the first definition of what it means to have location privacy in the presence of any general side information. For example, previous work [55, 35, 33, 24] suggests that clients should not upload data in sensitive or low-density areas or offers other heuristics, but provides no provable security guarantees and considers just specific side information. In contrast, we present two protocols (PD and SLP) that provide successively stronger privacy guarantees. With PD, clients can deny that they went on a certain path in a plausible way: with high probability, no one can prove that they actually followed a certain path unless the clients were physically observed. SLP achieves our location privacy definition: the aggregator gains no more information about the clients other than that provided by the aggregate result in the presence of any general SI. SLP requires the existence of a partially trusted lightweight intermediary (the *auxiliary*): even if malicious, it cannot change the statistics result; at worst, we do not have strict location privacy, but we fall back to plausible deniability. We will argue in section 6.2 that it is unlikely to be possible to provide strict location privacy without an auxiliary in our model.
2. *A protocol for accountability:* We describe in section 6.4 a solution to the problem of protecting against attempts by malicious participants to significantly bias the statistics by repeated false uploads. Our protocol allows the aggregator to enforce a quota on how much each client can upload at each path segment without knowing the identities of the clients and without using a trusted party. We introduce a novel cryptographic protocol that provides a decentralized solution to limited anonymous upload, based on an efficient zero-knowledge proof of knowledge that we design from scratch. This advance is significant because previous work either required clients to upload their id with the data [35], required impractical trusted parties [35] or simply offered no accountability [55] at all.

To validate our protocol and system design, we implemented the protocols and measured the performance of PrivStats. For the client, we used commodity hardware comparable to smartphone platforms. Both the upload protocols and the accountability protocols are fast, taking about 0.3 seconds to run. We also show that bandwidth usage and storage are reasonable and estimate that one commodity aggregator node could support the population of a large city. We conclude that PrivStats is easily practical with today's commodity smartphones and server hardware.

Chapter 2

Related Work

There has been a variety of related work in the area of protecting privacy in location-based mobile systems (vehicular or social). To illustrate clearly the contribution of each of our two systems, we present the related work individually for each and then compare them to each other.

2.1 VPriv's related work

VPriv is inspired by recent work on designing cryptographic protocols for vehicular applications [7, 8, 57]. These papers 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 secure multi-party computation as a black box, relying on general results from the literature for reducing arbitrary functions to secure protocols [63]. They do not provide actual protocols for the problems in question, and use the general result [26] that one can construct a secure multi-party protocol for any feasible function. However, if one follows the constructions from the literature, such protocols end up being very complex and slow. The state-of-the-art "general purpose" compiler for secure function evaluation, Fairplay [45], produces implementations which run more than three orders of magnitude slower than the VPriv protocol, and scale very poorly with the number of participating drivers (see Section 5.5.2). Given present hardware constraints, general purpose solutions for implementing secure computations are simply not viable for this kind of application. A key contribution of VPriv is to present a protocol for the *specific* class of cost functions on time-location pairs, which maintains privacy and is efficient enough to be run on practical devices and suitable for deployment.

Secure multi-party computation has also been used for maintaining privacy in location-based social applications [64]. This work solves the problem of determining if your friends are nearby without learning more information about their path. These protocols have been designed for the specific friend-finding problem, so they are practical. However, they are not applicable to the wider range of vehicular applications we are considering or, for example, to social games in which users collect points based on their path [23].

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 [15, 44, 4]. There has been a substantial amount of work on practical protocols for these problems so that they run efficiently on small devices (e.g., [12]). However, 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 more complicated congestion pricing. Moreover, physical attacks based on the details of the implementation and the associated bureaucratic structures remain a persistent problem [31]. 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, and we prove its efficiency empirically in Section 5.5.

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) [36, 40, 34]. The focus of this work is different from ours, although it can be used in conjunction with our work. These papers mostly attempt to compute traffic statistics while preserving privacy. They also analyze potential privacy violations associated with some specific side information at the server. VPriv does not prevent against side information leakage; it just ensures that the function computation does not leak any additional information besides what the server can gain from an anonymized database. These works conclude that it is possible to infer to what driver some GPS traces belong in regions of low density. In fact, PrivStats' goals are more similar to the goals of such work, and we present a comparison in the next subsection.

Using spatial analogues of the notion of *k-anonymity* [61], some work focused on using a trusted server to spatially and temporally distort locational services [33, 24]. 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 [37, 35], involving "cloaking" the data using spatial and temporal subsampling techniques. In addition, these papers [35, 37] 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 [35] 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 5.6 for further discussion of this point.

2.2 PrivStats' Related Work

Because the problem of protecting location privacy when computing aggregate statistics in mobile systems is a fundamental issue, there has been a substantial amount of prior work on this problem. As mentioned, early work focused on using a trusted server to spatially and temporarily alter location updates from clients [33, 24] or remove client identifier [35]. Some other approaches modify the data from the clients before releasing it [37].

One difficulty with these solutions is their reliance on fully trusted servers or intermediaries between the drivers and the aggregator. If these get compromised, so do the paths of the drivers (because the intermediaries have access to driver identifiers). Another problem is that simple inference algorithms can be used to recover detailed path information even from an anonymized database consisting of (time, location) pairs. Furthermore, such works either do not provide guarantees in the presence of side information attacks or they provide guarantees when treating only a subset of side information types and/or when assuming trusted parties. For example, in [37, 35], a trusted party aggregates the upload of each driver on a piece of road with the uploads of other $k-1$ drivers close in time and location to provide k-anonymity. While one individual tuple has k-anonymity, there is no proof that the driver is k-anonymous among other drivers over longer periods or that the driver's path is k-anonymous among other paths. The privacy guarantees are essentially heuristic and empirical in nature.

Our plausible deniability protocol provides a useful security property when there are no trusted parties. Even with trusted parties, none of the preceding work achieves the strong level of privacy we achieve in our strict privacy protocol. Moreover, the trusted party from the SLP protocol is just partially trusted: even when these parties are malicious, they cannot change the result of the statistics to be computed and the privacy will default to plausible deniability. In previous work, if one trusted party becomes malicious, the server will know mappings of client identifier to location and time (which essentially gives away a client's path) or will compute incorrect statistics; in our SLP protocol, in the worst case, the server will see only the anonymized database (with plausible deniability, in addition).

Moreover, accountability has previously either been ignored (allowing drivers to bias significantly the statistics), handled by having tuples contain driver identifiers (jeopardizing privacy), or relying on fully trusted parties. For instance, in [35], users upload tuples containing their unique ids to a trusted party that checks if clients have uploaded too much.

Work on e-cash [13] is related to our accountability protocol (Section 6.4): one might envision giving each client a number of e-coins that they can spend for every road. This approach is used with success in spam control applications [3, 62]. However, it is not practical in our setting: the coins must be tied to the particular statistics, location, and interval, which requires a prohibitively large number of coins. Moreover, e-cash adds a lot of complexity and overhead for identifying double-spenders, which is not needed in our setting.

Finally, we remark as with VPriv, that our approach is complementary to the work on differential privacy [18]. Such work can guide PrivStats in deciding what aggregate statistics should not be known to the aggregator (e.g., statistics in low-density areas).

2.3 VPriv versus PrivStats

VPriv and PrivStats address different problems in location privacy: how to compute functions over a particular individual's path versus how to compute privately aggregate statistics over all the clients. VPriv associates the result of a function to a client's id, whereas PrivStats keeps each client's contribution to the statistics anonymous from the aggregator (in addition to the client's path). In VPriv, clients can upload as many tuples as they wish, but they are not allowed to not upload; in PrivStats, clients can decide not to upload, but they must be prevented from uploading too much. Because of these sets of different goals, VPriv did not seem applicable to the aggregate statistics problem in PrivStats; PrivStats necessitated markedly different cryptographic protocol designs.

Besides solving a different problem, PrivStats also furthers VPriv by providing solutions to problems that were open in VPriv (and related work) such as side information. As discussed, PrivStats provides a definition of privacy for general side information attacks and a protocol that delivers to this definition (as well as the plausible deniability protocol which provides a useful security property). VPriv can also be adapted to benefit from our protocols against side information attacks.

Chapter 3

Cryptographic Background

The work in this thesis employs various cryptographic concepts and tools. Unless otherwise noted, the notation and formal definitions in this section are based on [25], which is also an excellent source of further details and related notions. In this thesis, we make standard cryptographic assumptions that have been widely used in the literature (e.g., the strong-RSA assumption).

3.1 Basics

Encryption and signature schemes lie at the foundations of our protocols.

3.1.1 Encryption Schemes

Encryption allows private information exchange between two parties, a sender (called Alice) and a receiver (called Bob). The channel of communication between these two parties is unreliable and may allow a malicious adversary to eavesdrop messages exchanged. Encrypting messages sent over this channel prevents the adversary from learning their content. The following is a formal definition:

Definition 1 (Encryption Scheme)**.** *An encryption scheme is a triple, (G, E, D), of probabilistic polynomial time algorithms satisfying the following two conditions:*

1. *(Key generation.) On input 1^n, algorithm G (called the key-generator), outputs a pair of bit strings.*
2. *(Encryption/Decryption.) For every pair (e, d) in the range of $G(1^n)$, and for every $\alpha \in \{0,1\}^*$, algorithms E(encryption) and D(decryption) satisfy $\Pr[D(d, E(e, \alpha)) = \alpha] = 1$, where the probability is taken over the internal coin tosses of algorithms E and D.*

The standard desired security property of encryption schemes is semantic security (see [25] for more details).

A **private-key encryption scheme** (symmetric encryption) is an encryption scheme in which the sender and the receiver previously agreed upon a secret key, which they use for encryption and decryption.

A **public-key encryption scheme** (asymmetric encryption) is an encryption scheme in which the encryption key differs from the decryption key. Moreover, the encryption key is publicly known to everyone and it is infeasible to find the decryption key from the encryption key.

3.1.2 Signature Schemes

A **signature scheme** is a method for verifying that the data is authentic; that is, it comes from an approved party. Alice can sign any messages she sends to Bob and Bob can verify whether a message he receives is indeed from Alice by verifying the signature.

Definition 2 (Signature Scheme)**.** *A signature scheme is a triple (G, S, V) of probabilistic polynomial-time algorithms satisfying the following two conditions:*

1. *On input 1^n, algorithm G (called the key-generator) outputs a pair of bit strings.*

2. *For every pair (s, v) in the range of $G(1^n)$, and for every $\alpha \in \{0,1\}^*$, algorithms S (signing) and V (verification) satisfy* $\Pr[V(v, \alpha, S(s, \alpha)) = 1] = 1$*, where the probability is taken over the internal coin tosses of S and V.*

The standard definitions of security for signature schemes are existential unforgeability under chosen message attack, also detailed in [25].

3.2 Cryptographic Tools

This thesis makes use of a variety of cryptographic tools, each coming with some useful security guarantees and a model of functionality.

3.2.1 Commitment Scheme

A **commitment scheme** [11] consists of two algorithms, *Commit* and *Reveal*(or decommit). 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 (she needs to have "committed"). This means that, later when she wants or has to reveal v to Bob, she cannot provide a different value, $v' \neq v$, which convinces Bob that was the value committed (that is, that it corresponds to the ciphertext provided by Alice). Specifically, to commit to v, Alice computes $Commit(v) \rightarrow (c_v, d_v)$, where c_v is the resulting ciphertext and d_v is a decommitment key; c_v and d_v have the following properties:

- Bob cannot gain any information about v from c_v.

- Alice cannot provide $v' \neq v$ such that $Commit(v') \rightarrow (c_v, d')$, for some d', subject to standard computational hardness assumptions.

We say that Alice reveals v to Bob if she provides v and d_v, the decommitment value, to Bob, who already has c_v.

In our protocol, Bob (the server in VPriv or aggregator in PrivStats) will store all commitments he receives from Alice (the client), and Alice will store all decommitment keys. Therefore, we use the notation $c(v)$ and $d(v)$ to mean the commitment corresponding to v and the decommitment corresponding to the commitment to v.

3.2.2 Pseudorandom functions

A pseudorandom function family [49] 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 (under standard computational hardness assumptions).

3.2.3 Homomorphic encryption

Homomorphic encryption is an encryption scheme in which performing an operation on the encryptions of two values (v_1 and v_2) results in an encryption of values combined with a second operation. For example, RSA [2] is a multiplicative homomorphic encryption scheme because multiplying two ciphertexts results in a ciphertext of the underlying plaintexts multiplied.

In VPriv, we use a **homomorphic commitment** scheme (such as the one introduced by Pedersen [53]). Pedersen's scheme has the property that $c(v) \cdot c(v') = c(v + v')$ and $d(v + v') = d(v) \cdot d(v')$. If Alice committed to v and v' and wants to reveal $v + v'$ alone to Bob, she will give him $v + v'$ accompanied by $d(v + v')$. Bob can compute $c(v + v')$ by multiplying the commitments to v and v' he received from Alice.

3.2.4 Blind signature scheme

In PrivStats, we use a blind signature scheme: it is a signature scheme in which the signer manages to sign a message without seeing the message signed or the signature produced. Nevertheless, the signer can control how many signatures he produces for a certain client.

3.3 Zero-knowledge notions

3.3.1 Secure multi-party computation

A secure multi-party computation [63] is a protocol in which several parties holding private data interact to compute a function on everyone's private data. At the end of the protocol, if at least a majority of the parties were honest, the correct result is obtained. Moreover, none of the participants can learn the private information of any other party beyond what can be inferred from the result of the function.

3.3.2 Zero-knowledge proofs and proofs of knowledge

A zero-knowledge proof [29], at a high level, is a proof that involves proving the truth of a statement without revealing any information other than the validity of the statement.

Proofs of knowledge [30], [5], [60] are proofs by which a machine can prove that it knows some value that satisfies a certain relation. For example, Schnorr [60] provided a simple and efficient algorithm for proving possession of a discrete log. If we add the zero-knowledge [30] property, we also have the guarantee that no information about the value in question is leaked.

For a formal definition of zero-knowledge proofs of knowledge, which we will use in our proofs in the appendix, we use the definition in [46].

Consider a predicate $Q : \{0,1\}^* \times \{0,1\}^* \to \{\mathsf{false}, \mathsf{true}\}$. For a given bit-string z, a prover claims to know a bit-string x with $Q(z,x) = \mathsf{true}$.

Definition 3. *An interactive protocol (Prover, Verifier) is a zero-knowledge proof of knowledge for predicate Q if the following holds:*

- *Completeness: Verifier accepts when Prover has as input an s with $Q(z,x) = \mathsf{true}$.*
- *Proof of knowledge: There is an efficient program K, called knowledge extractor, with the following property. For any (possibly dishonest) prover $\hat{P}$ with non-negligible probability of making Verifier accept, K can interact with $\hat{P}$ and outputs (with overwhelming probability) an x such that $Q(z,x) = \mathsf{true}$.*
- *Zero-knowledge: For every efficient verifier $\hat{V}$, there exists a simulator S, such that the output of S is indistinguishable from a transcript of the protocol execution between Prover and $\hat{V}$.*

PrivStats uses the blind signature scheme from [14], which is accompanied by a proof of knowledge of a signature for a value that is hidden inside a given commitment.

3.3.3 Discussion

Secure multi-party computation and zero knowledge proofs of knowledge are notions of security and not constructions. One can design protocols for various problems and then prove that they satisfy these security definitions. Goldreich et Al. showed in two seminal papers ([27], [26]) that any language in NP has a zero-knowledge proof and that any game can be transformed in a secure multi-party computation if a majority of the parties are honest. Intuitively, this means that we can come up with secure multi-party computation protocols and zero-knowledge proofs for most protocols encountered in practice. Moreover, these papers as well as others in the literature provide general constructions for any feasible problem based on reductions to certain problems with known protocols. However, using such constructions in practice results in prohibitively slow and complex protocols. In practice, one should design such secure protocols by exploiting the specifics of the problem at hand and then prove it has the desired zero-knowledge properties. This can result in very efficient protocols; in particular, VPriv relies on such a zero-knowledge proof and is three orders of magnitude faster than the general reductions, as we will explain in Chapter 5.5.2.

Chapter 4

Model

In this section, we describe the framework underlying our schemes, our goals, and the threat models we consider.

4.1 General Model

Both VPriv and PrivStats rely on a similar and general framework. There are two types of participants: many clients and a logical server. Clients can be drivers in location-based vehicular services, peers in a social network, or any other participants in a location-based service. The server is an entity interested in computing a function or some statistics on the drivers' paths. The model requirements are:

- Clients can upload time-location information to a remote server using some portable device
- Clients have some minimal computation and storage abilities
- Clients can communicate with a server
- Information can be taken in and out of the device

This framework is fairly general and captures a broad class of vehicular and social location-based services. We elaborate on what specific devices clients can use in these systems in the next subsections.

A first step for privacy that both systems implement is that clients do not upload their identifiers at the server when uploading data.

In both systems, the basic threat model is similar. The server is trusted to perform the computation correctly, but he is not trusted with the privacy of the clients: it would like to learn clients' paths and may leak such data. The clients may try to bias the computation or statistics result in their favor.

Since VPriv and PrivStats have different computation goals, they have different instantiations of this more general model.

4.1.1 VPriv's Framework

In VPriv, clients are typically drivers. Drivers operate cars, cars are equipped with transponders or smartphones that transmit information to the server, and drivers also run *client software* which enacts the cryptographic protocol on their behalf.

The server computes some function f (e.g., tolling, law violation, insurance calculations) 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 computes the function. 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. Thus, 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 3.2.1, 5.2): 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, and speed of the car. For example, we might wish to assess higher premiums on cars that persistently drive close to the speed limit, or are operated predominantly late at night.

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. In fact, our general framework can be applied to any function over path tuples because of the general result that every polynomially computable function has a secure multi-party protocol [29, 63, 27, 26]. However, as discussed in Section 5.5.2, 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 (transponder may be tampered with) 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 [21]) or they can be batched until the end of the day or of the month. Section 5.6 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 [38]), toll transponder systems such as E-ZPass [32], and roadside surveillance systems.

4.1.2 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 5.6. 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 (e.g. does not claim the driver failed a verification test, when she did not). We believe this is a reasonable assumption 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 5.6, the protocol can be made resilient to a fully malicious server as well with very few modifications.

4.1.3 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 private 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 as follows:

Definition 4. ***(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.*
- $\mathcal{V}$ *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 channels present in the raw database* $\mathbb{S}'$*.*

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.

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 and receives (from an oracle) the result of the function (without running any protocol). 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 certain roads at a particular time, then that car's privacy will be violated. See Section 5.6 for further discussion of this issue.

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.

Note that we do not aim to hide the result of the function; rather, we want to compute this result without revealing private information. In some cases, such as tolling, the result may reveal information about the path of the driver. For example, a certain toll cost may be possible only by a combination of certain items. However, if the toll period is large enough, there may be multiple combinations of tolls that add to the result. Also, finding such a combination is equivalent to the subset-sum problem, which is NP-complete.

4.2 PrivStats' Framework

4.2.1 Setting

In our model, we have two parties. The *clients* are mobile devices with wireless connectivity; the *aggregator* is an entity interested in computing certain aggregate statistics over the clients.

In our SLP protocol, we will need a third party, the *auxiliary*. This is a potentially distributed entity which receives uploads from the clients and delivers them to the aggregator.

The client must be able to communicate with the aggregator and the auxiliaries over its wireless network. The network connectivity could be continuous or intermittent. We do not make any assumptions about the client, and the aggregator cannot assume any knowledge of what software is running on the client. The aggregator also has no recourse to ensure participation in any portion of the protocol.

The aggregator is interested in computing some aggregate statistical function periodically from the uploaded data. These statistics are calculated for various location-time combinations known in advance (e.g. published in an online database); for example, the average speed on road segment S (between blocks S_1 and S_2) for time in the range $T_1 =$ 4.00 pm to $T_2 =$ 4.15 pm. We use the term *pointstamp* to refer to the combination of location and time slot; the aggregator can compute one or more statistical functions for any given pointstamp. We assume that the clients know a priori from the aggregator which statistical functions will be computed at any given pointstamp. The aggregator may choose to compute the function in "real time", or may choose to do so at a later point in time. Tuples that have been uploaded do not have to be discarded by the aggregator to provide location privacy even when an audit of the uploaded data is conducted. To upload tuples to the aggregator, we assume that clients use an anonymizing network such as [17], a proxy aggregator, or any other method that hides IP address information.

A client periodically logs tuples of the following form:

$$\langle \mathsf{attribute}, \mathsf{value}, \mathsf{pointstamp}, \mathsf{token} \rangle \tag{4.1}$$

The attribute refers to the type of data (aggregation function) whose value is being logged, such as "speed", "acceleration", etc. The client sends logged data to the aggregator from time to time, as long as the privacy protocol says that it is fine to do so for any given pointstamp. The token in 4.1 is used in the accountability protocol to verify that the upload is legitimately within an assigned quota. As we will see in Section 6.4, the token does not reveal any information about the identity of each client and changes from one upload to another.

Since attribute and pointstamp uniquely identify a statistics to be computed, we can think of them as a statistic ID or statID (statID could be a concatenation or a hash of the two fields). Client can also just upload statID instead of the two fields, if the mapping is publicly known.

4.2.2 Threat Model

We assume that the aggregator cannot be trusted to protect the privacy of participants: the aggregator might attempt to recover the path of a given client and release this information to third parties without the consent or knowledge of the participant (e.g., for advertising or more nefarious purposes). However, the aggregator is trusted to compute the aggregate statistics correctly; we assume that this is in fact what the aggregator wants to compute.

On the other hand, the clients cannot be trusted to correctly upload values. Unlike in tolling applications (e.g., the scenario considered in [55]), the aggregator is assumed to have no ability to compel compliance of the clients. Even honest clients might suddenly stop participating in the protocol. Clients might malfunction in various ways, and malicious behavior is also possible, as clients may have incentives to try to bias the statistics (depending on the specific application). Specifically, since we will require uploads to be anonymous, we have to address attacks by malicious nodes in which large numbers of bad tuples are uploaded.

We assume the auxiliaries are potentially untrusted; we provide different privacy guarantees depending on the behavior of the auxiliaries.

4.2.3 Goals

In light of the threats outlined above, we identify the following desiderata for our system. Our goal is to compute aggregate statistics over the client's paths such that:

1. The aggregator or other parties with access to the complete history of uploads cannot learn private location information about the clients beyond what is explicitly permitted as part of the aggregate statistics contribution. This property should be preserved even in the face of side information.

2. A malicious or damaged client will have quantifiable impact on the aggregate statistics, with limited bias introduced unless the overall rate of uploads is low.

3. The aggregate statistics can be computed within a short time after the "collection interval" ends. This requirement arises because we wish to obtain traffic information in "real time" to guide route planning.

4. Client participation in a "reconciliation phase" must not be necessary for computation of the statistics (which rules out the kind of solutions employed in VPriv [55]).

5. No global trusted third-party is required for the protocol; any intermediaries used must be untrusted and the consequences of their breakdown or corruption must be limited.

6. Clients cannot reliably directly communicate with other nearby clients; we regard this as impractical in the vehicular setting.

7. The protocol is efficient enough to run on commodity hardware for the aggregator and auxiliary and on a commodity smartphone for the client.

We will explain how PrivStats achieves all these goals.

Chapter 5

VPriv

The backbone of surprise is fusing speed with secrecy.
– Karl Von Clausewitz, Prussian historian and military theorist

In this chapter, we present VPriv, our system for computing functions over the paths of drivers while preserving their privacy.

5.1 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 4.1.1. 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 3.2.1 and 5.2. 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 5.2). The server *does not know* which car uploaded a certain tuple. To ensure that the transponder

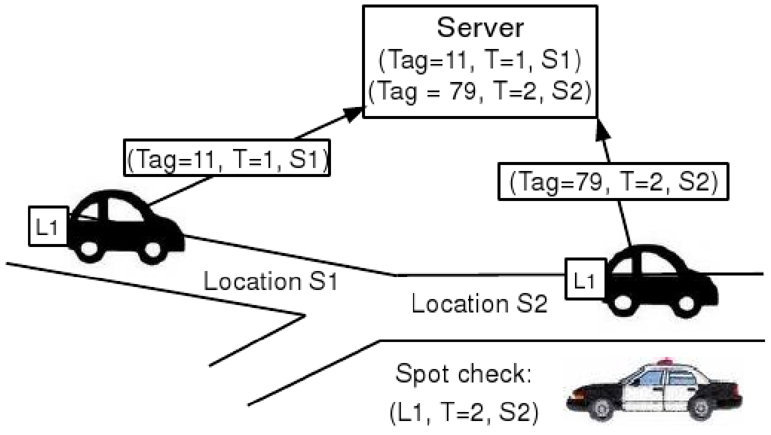


Figure 5-1: Driving phase overview: A car with license plate L1 is traveling from Location S1 at time 1 to Location S2 at time 2 when it undergoes a spot check. It uploads path tuples to the server.

abides by the protocol, VPriv also uses sporadic random spot checks that observe the physical locations of cars, as described in Section 5.3. 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. Figure 5-1 illustrates the driving phase.

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. He does not need to transfer any information from the transponder to the computer (unless the tuples can be corrupted or lost on their way to the server and the client needs to check that they are all there). It is enough if his computer knows the initial tags (from registration). 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 5.3). 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.*

$COST$	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, $Commit(x) = (c(x), d(x))$.
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 formed 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 5-2: Notation.

The reconciliation can be done transparently to the user the client software; from the perspective of the user, he only needs to perform an online payment.

To implement this protocol, VPriv uses a set of modern cryptographic tools: a homomorphic commitment scheme and random function families. They are described in Chapter 3.

5.2 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.

5.2.1 Tolling protocol

We first introduce the notation in Figure 5-2. For clarity, we present the protocol in a schematic manner in Figures 5-3 and 5-4. The protocol is illustrated for only one round for simplicity. For multiple rounds, we need *a different random function for each round.* (The reason is that if the same random function is used across rounds, the server could guess the tuples of the driver by posing a $b = 0$ and a $b = 1$ challenge.) The registration phase is the same for multiple rounds, with the exception that multiple random functions are chosen in Step (a) and Steps (b) and (c) are executed for each random function.

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. Performing both means figuring out the driver's tuples.

1. **Registration phase:**
   (a) Each client chooses random vehicle tags, v_i, and a random function, f_k (one per round), 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** (client proves that $COST$ is correct) begins.

Figure 5-3: VPriv's protocol for computing the path tolling cost (small modifications of this basic protocol work for the other applications). The round protocol is presented in Figure 5-4.

These verifications and computations occur within a round, and there are multiple rounds. During each round, 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 result. 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. The number of rounds is fixed and during registration the client selects a pseudorandom function f_k for each round 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 $b = 1$). We do not regard this as a significant problem, since the very fact that a particular amount was paid may reveal this number (especially for cases where the tolls are about equal). However, if desired, we can handle this problem by uploading some "junk tuples". These tuples still use valid driver tags, but the location or time can be an indication to the server that they are junk and thus the server assigns a zero

The round protocol

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 $(b = 1)$ 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 all 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, $I = \{v_i, \forall\ i\} \cap \{s_j, \forall\ j\}$. 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 3.2.3, 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 5-4: The protocol executed during each round in the reconciliation phase. The arrows indicate data flow.

cost. These tuples will be included in the tolling protocol when the server will see them encrypted and will not know how many junk tuples are in the intersection of server and driver tuples and thus will not know how many actual tolling tuples the driver has. Further details of this scheme are not treated here due to space considerations.

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 Figures 5-3 and 5-4 results in the correct tolling cost and respects the driver's location privacy.*

Please find the proof in the appendix.

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 5.4.

Point tolls (replacement of toll booths). 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 tag; we discuss this refinement in detail for computation of speed below in Section 5.2.2. 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.

5.2.2 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 it is straightforward to extend the solution to varying speed limits. 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; 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, else it is zero. 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.

Note that this protocol leaves "gaps" in coverage during which speeding violations are not detected. Since these occur every other upload period, it is hard to imagine a realistic driver exploiting this. Likely, the driver will be traveling over the speed limit for the duration of several tuple creations. However, if this is deemed to be a concern for a given application, a variant can be used in which the period of changing tuples is divided and linked pairs are interleaved so that the whole time range is covered: $\ldots v_2\ v_1\ v_3\ v_2\ v_4\ v_3\ v_5\ v_4\ v_6\ v_5 \ldots$

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.

5.2.3 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 violation of local speed limits.

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, in fact the exact framework of the speeding ticket example will suffice.

5.3 Enforcement

The cryptographic protocol described in Section 5.2 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 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`$\rangle$. We show in Section 5.5 that such spot checks can be infrequent (and thus do not affect driver privacy), while being effective.

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 performed by secret cameras that are able to take pictures of the license plates. At the end of the day or month, an officer could extract license plate, time and location information or this task could be automated. Alternatively, using the existing surveillance infrastructure, spot checks can be carried out by roving police cars that secretly record the car information. This is similar to today's "speed traps" and the detection probability should be the same for the same number of spot checks.

The data from the spot check is then used to validate the entries in the server database. In the reconciliation phase of the protocol from Section 5.2, the driver is also required to prove that she uploaded a tuple that is sufficiently close to the 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 tuple (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 claim that another driver's tuple belongs to his due to the commitment check. The server may allow a threshold number of tuples to be missing in the database to make up for accidental errors. Before starting the protocol, a driver can check if all his tuples were received at the server and upload any missing ones.

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 5.5, 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 driving phase with tuples from registration. 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 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.

5.4 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. We implemented the tolling protocol from Figure 5-3, where we used the Pedersen commitment scheme [53] and the random function family in [49], 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 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.

5.4.1 Downloading a subset of the server's database

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, 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. For correctness, the client needs to prove that all of her tags are among the ones downloaded. Let the number of encrypted tags provided to the server during registration be n; the first $m \leq n$ of these tags have been used in the last reconciliation period. Assume the driver informs the server of m. Any misreporting regarding m can be discovered by the enforcement scheme (because any tags committed to during registration but not included in the first m will not verify the spot check). When step (iv) is executed for $b = 1$, the server also checks that all the first m tuples are included in the set s_i; that is $\{f_k(v_i) | i \leq m\} \in \{f_k(s_j) | \forall j\}$.

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 intervals. 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.

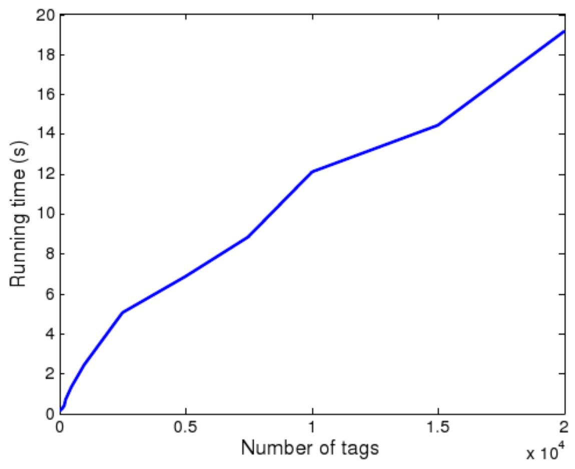


Figure 5-5: The running time of the road pricing protocol as a function of the number of tags generated during registration for one round.

Alternatively, during registration clients could be assigned random "tag subsets" which are then subsequently used to download clusters of tags; the number of clients per tag subset can be adjusted to achieve the desired efficiency/ privacy characteristics. The tag subset could be enforced by having the clients pick random tags with a certain prefix. Clients living in the same area would belong to the same tag subset. In this way, a driver's privacy comes from the fact that the server will not know whether the driver's tuples belong to him or to any other driver from that region (beyond any side information).

5.5 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 5.5.3 and with real data traces in Section 5.5.3.

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 round 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 5.4. We are concerned primarily with measuring the cryptographic overhead.

5.5.1 Execution time

Figures 5-5, 5-6, and 5-7 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

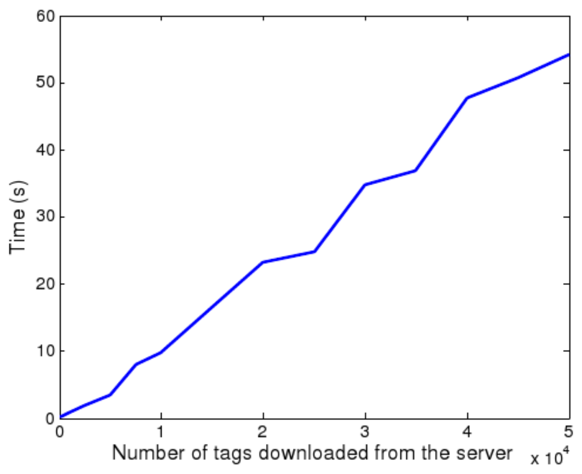


Figure 5-6: The running time of the road pricing protocol as a function of the number of tuples downloaded from the server during the reconciliation phase for one round.

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 minute. This interval is adjustable; the 1 minute seems reasonable given the 43 MPH average speed [52]. The average number of miles per car per year in the US is $14,500$ miles and 55 min per day ([52]), which means that each month sees about ≈ 28 hours of driving per car. Picking a new tag once per minute leads to $28 \times 60 = 1680$ tags per car per month (one month is the reconciliation period that makes sense for our applications). So a car will use about 2000 tags per month.

We consider that downloading $10,000$ tuples from the server offers good privacy, while increasing efficiency (note that these are only tuples with non-zero tolling cost). The reason is as follows. A person roughly drives through less than 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 with the 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 5-7. 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 [7] 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 times 86400 seconds per day divided by 54 seconds per car = 51840 cars per month. Even if bandwidth does not scale linearly with the

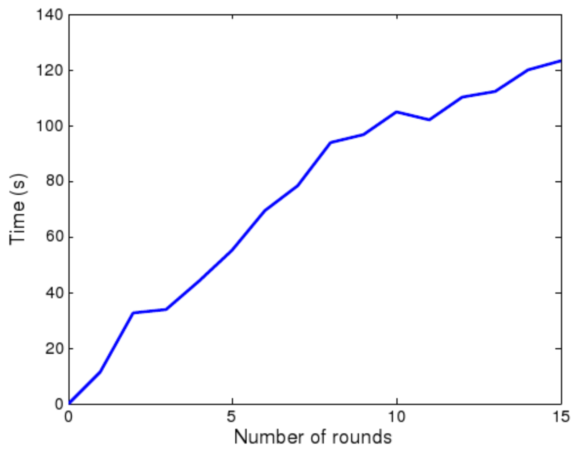


Figure 5-7: The running time of the road pricing protocol as a function of the number of rounds used in the protocol. The number of tags the car uses is 2000 and the number of tuples downloaded from the server is 10000.

number of cores, the latency due to bandwidth utilization is still one order of magnitude less than the one for computation; even if it adds up and cannot be parallelized, the needed number of cores is still within the same order of magnitude. Also, several computers can be placed in different parts of the network in order to parallelize the use of wide-area bandwidth. Since the downloaded content for drivers in the same area is the same, a proxy in certain regions will decrease bandwidth usage significantly. Hence, for 1 million cars, one needs $10^6/51840 \approx 21 < 30$ similar cores; this computation suggests our protocol is feasible for real deployment. (We assumed equal workloads per core because each core serves about 50000 users so the variance among cores is made small.)

5.5.2 Comparison to Fairplay

Fairplay [45] is a general-purpose compiler for producing secure two-party protocols that implement arbitrary functions. It generates circuits using Yao's classic work on secure two-party computation [63]. We implemented a simplified version of the tolling protocol in Fairplay. The driver has a set of tuples and the server simply computes the sum of the costs of some of these tuples. We made such simplifications because the Fairplay protocol was prohibitively slow with a more similar protocol to ours. Also, in our implementation, the Fairplay server has no private state (to match our setting in which the private state is only on the client). 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 75 tags, and compiling and running the protocol in such a case required over 5 minutes. In comparison, our protocol runs with about $10,000$ tuples downloaded from the server in 100s, which yields a difference in performance of *three 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 which significantly improve its performance and bandwidth requirements; notably, the use of ordered binary decision diagrams [41]. However, the performance improvements associated with this work are less than an order of magnitude at best, and so do not substantially change the general conclusion that the general-purpose implementation of the relevant protocol is orders of magnitude slower than VPriv. This unfeasibility of using existing general frameworks required us to invent our own protocol for cost functions over path tuples that is efficient and provides the same security guarantees as the general protocols.

5.5.3 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.

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 (or similarly, driving through a segment). 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

$$\Pr[\text{spot check}] = 1 - (1-p)^m. \tag{5.1}$$

Figure 5-8 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 $p = 1/500$. In this case, each car has an expected time of 500 minutes (8.3h) of driving until it undergoes a spot check and will be observed with 95% probability after about 598 min ($<$ 10 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 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 much smaller number of spot checks will suffice. 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 from randomized law enforcement suggests strongly that independent of β, Pr[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) [19].

If there is concern about the possibility of tuples lost in transit from client to server, our protocol can be augmented with an anonymized interaction in which a client checks to see if all of her tuples are included in the server's database (the client can perform this check after downloading the desired tuples from the server and before the spot check reconciliation and zero-knowledge protocol). Alternatively, the client might simply blindly upload duplicates of all her tuples at various points throughout the month to ensure redundant inclusion in the database. Note that it is essential that this interaction should be desynchronized from the reconciliation process in order to prevent linkage and associated privacy violation.

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 $1-\sum_{i=0}^{t}\binom{m}{i}p^i(1-p)^{m-i} \geq 1-e^{\frac{-(t-mp)^2}{2mp}}$, where the last inequality uses Chernoff bounds.

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 [38], 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 spend fuel and time resources by driving through these roads and which most likely do not even have tolls. More sophisticated placement schemes are possible; here, we are primarily

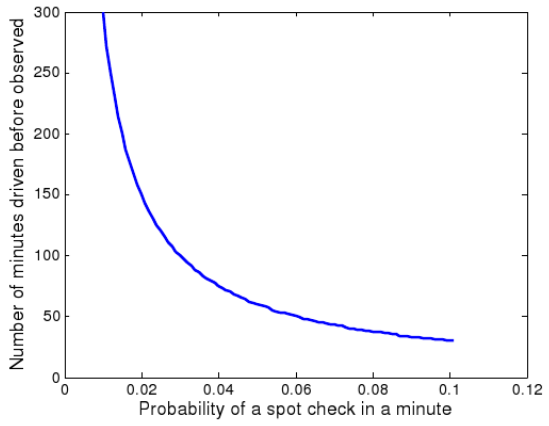


Figure 5-8: The time a motorist can drive illegally before it undergoes a spot check with a probability 95% for various values of p, the probability a driver undergoes a spot check in a minute.

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% ($\approx$ 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 5-9 illustrates the data obtained. In few places, the graph is not perfectly monotonic and this is due to randomization: we are placing few spot checks in some of the 300 locations. Even if in some cases we place a spot check more than in others, due to randomization, the spot checks may be placed in an unfavorable position and observe less paths. The reason is that the 300 spot check vary significantly in popularity. From the shape of the graph, 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 5.5.3. After 10 spot checks have been placed, the fraction of paths observed grows much slower. This is because we are only placing spot checks at 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 5.5.3, the number of spot checks increases logarithmically in the probability of detection of each driver and thus the percentage of drivers observed.

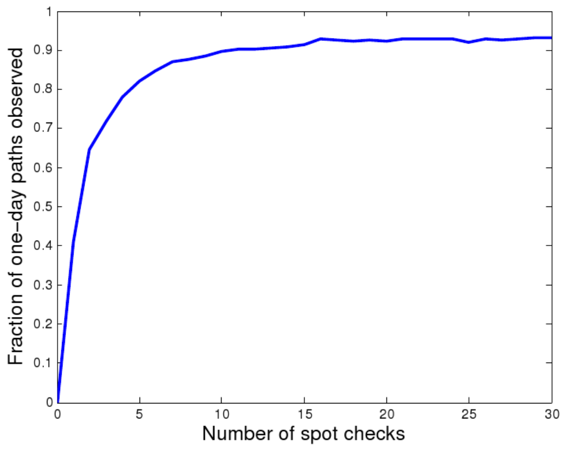


Figure 5-9: The fraction of one-day paths observed out of a total of 4826 one-day paths as a function of the total number of police cars placed.

5.6 Security analysis

In this section, we discuss the resistance of our protocol to the various attacks outlined in Section 4.1.2.

Client and intermediate router attacks. Provided that the client's tuples are successfully and honestly uploaded at the server, the analysis of Section 5.2 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. To deal with dropped or forged tuples, 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 5.2 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 [17], or onion routing [28]. The client can avoid any timing coincidence by sending these tuples in separate packets (perhaps even at some intervals of time) towards the end of the driving period, when other people are sending such tuples.

Another issue is the presence of side channels in the anonymized tuple database. As discussed in Section 2.1, a number of papers have demonstrated that in low-density regions it is possible to reconstruct paths with some accuracy from anonymized traces [36, 40, 34]. As formalized in Definition 4, our goal here 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 [37, 35], which use a *time to confusion* metric that measures how long it takes an identified vehicle to mix back into traffic. In [35], 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 5.2, 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 5-3, 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. This will prevent the server from changing the client's ciphertext because he cannot forge the client's signature.

- 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 (e.g., 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.

Chapter 6

PrivStats

Too many people are thinking of security instead of opportunity.
– James F. Byrnes, American statesman

In this chapter, we present PrivStats, our system for computing aggregate statistics over the paths of drivers while preserving their privacy.

6.1 PrivStats at High Level

In this section, we provide an overview of PrivStats, before we delve into the details of its protocols.

Recall that each tuple uploaded by a client is anonymized insofar as there is no identifier linking the tuple to the client. However, as discussed, a great deal of information about paths can be recovered from such anonymized tuple databases, and in the presence of side information, recovered paths can be linked to specific drivers. Furthermore, there is nothing to prevent a malicious client from uploading arbitrary quantities of bad data.

Our first protocol achieves *plausible deniability* and *accountability*; even in the presence of side information, the aggregator cannot link a particular path to a driver in a provable way, and drivers can only upload limited quantities of data while preserving anonymity. The high level idea is that each client will upload data with a certain probability at each pointstamp. We discuss the mechanism for selectively not uploading in Section 6.2 and the functioning of the "quota token" in Section 6.4. Roughly speaking, the upload policy ensures that the aggregator cannot tell which client uploaded a given tuple short of direct observation.

However, this protocol still suffers from the generic problems with maintaining such tuple databases; paths can be recovered in sparse areas. Our second protocol requires more infrastructure, but provides *strict locational privacy*; the aggregator receives no more information than that afforded by the computed statistic.

6.2 Aggregation Protocols

In this section, we discuss and analyze various threats to location privacy via side information, formalize various kinds of anonymity guarantees, and present the aggregate statistics

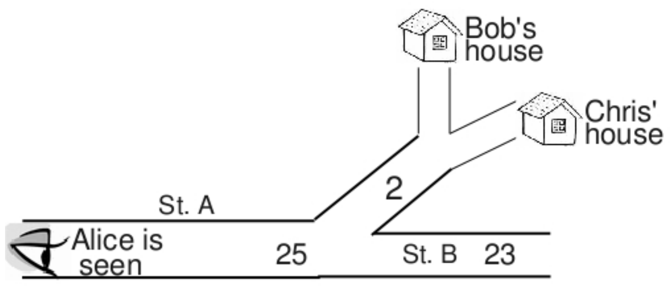


Figure 6-1: Example of path inference of a client using side information about other client's. The number of the road indicates the number of tuples uploaded from that road. The side information at the aggregator is that Alice, Bob and Chris were together on street A and where Bob's and Chris' houses are located. Therefore, the aggregator can deduce that Alice went from street A to street B.

protocols. We focus here on location privacy; in Section 6.4 we discuss how to handle malicious clients.

6.2.1 Side information attacks

Tuples in PrivStats do not contain any identification information about the client, but this level of anonymization hardly suffices to protect location privacy. Previous work [40] shows that it is possible to recover significant path information from anonymized time-location tuples. Furthermore, if the aggregator has side information specific to a client, it can use this information in concert with the tuples received to learn further information about the client as explained below.

- **Areas of low density.** Assume that the aggregator knows that Alice alone lives on a particular street. If the aggregator receives one tuple from that street, the aggregator can conclude that Alice just left her house or returned to it.
- **Side information about other clients.** Figure 6-1 illustrates an example.
- **Physical observation.** A client is observed at a specific location and time.
- **Others.** One can combine knowledge of the map of a city, approximate flow conservation, and driving patterns to infer paths.

Our goal is to first define location privacy in the presence of side information, and then to develop protocols that can provide it.

6.2.2 Location privacy definition

Side information can come in different forms and reveal various degrees of information. Clearly, one cannot hope to enumerate all the types of side information there can be, or know in advance what privacy leakage will result. This makes it challenging to provide theoretical guarantees in the face of side information. For example, one can never guarantee that Alice's path between point A and point B will remain unknown because Alice may simply be observed at some location.

We use the term "adversary" to refer to a curious aggregator (one that wants to learn location information about clients) or more generally to any party that gets data from the aggregator (e.g. a court via a subpoena) and wants to learn Alice's path.

So what does it mean for a protocol to guarantee location privacy? Obviously, we cannot prevent the adversary from having side information. Instead, the idea is that the protocol should not reveal any *additional* information about the path of Alice beyond what the aggregator already knows and the result of the statistics. (The result of the statistics is the goal of our computation so we want it to be public.) Of course, one can use differential privacy approaches [18] with PrivStats to decide whether some results should be released at all.

Definition 5. ***Strict location privacy (SLP).*** *Let SI be the side information available to the adversary, Result be the result of the aggregations to be computed, Data be the collection of all tuples uploaded at the aggregator when running a protocol. We say that the protocol maintains location privacy if the probability that the adversary guesses Alice's path given SI and Result is the same as the probability of guessing Alice's path when the server is additionally given Data.*

The preceding definition can be formalized via standard notions in cryptography: In cryptographic terminology, the difference between the chance of success of the adversary in the case when he is given a random collection of tuples values and the actual tuples is insignificant (and no bounded polynomial time algorithm can exploit such difference).

Intuitively, this definition says that the database of tuples used in our protocol to compute the desired result leaks no location privacy information.

6.2.3 Plausible deniability (PD)

Before we present a protocol that provides SLP, we present a protocol, PD, that is simple and lightweight and may be preferred in some cases in practice. PD does not provide as strong security properties as SLP, yet it has some interesting security property that may suffice in some cases.

Alice has *plausible deniability* if she is able to credibly deny that she went on a path (on which she actually drove). A motivating real example [1] of the utility of such a guarantee is when, due to a subpoena, the aggregator is forced to release the tuple database. In the face of a specific accusation, the client should be able to deny that it went on a particular road.

We first make the assumption that any tuple uploaded from a street could have originated from any client driving on that street; that is, the tuple's statistics value does not help distinguish between potential uploaders. This should hold true for most statistics (e.g. average speed, delay, others). Recall that tuples never contain identification in PrivStats.

Then, observe that the number of tuples uploaded from a road (coupled with side information) can jeopardize a client's plausible deniability. For example, consider that Alice and Bob are the only clients uploading from a certain street. If there are two distinct uploads from the same street at the same time, it is almost certainly the case that both Alice and Bob drove on that street at that time.

We achieve plausible deniability using a simple, yet interestingly powerful observation. Plausible deniability can be achieved if there are always some clients who refrain from uploading at each statistics point. In this way, for whatever tuples Alice uploads, she can claim that those tuples were uploaded by some other driver; this second driver is one of the drivers who did not upload at the statistic point. Assuming all tuples could have been generated by any driver, the tuples received at the aggregator are the same as in the case when Alice never drove through that path and a driver from the ones refraining uploaded in her case. Returning to the example of Alice and Bob, if one of them refrains from uploading, Alice can claim that Bob uploaded the tuple and Bob can claim that Alice did so and no one can disprove their claim without direct observation, of course.

How can we ensure that some nodes do not upload? As mention in Section 4.2, we cannot assume that clients can communicate and synchronize. The idea is to have clients choose probabilistically whether to upload. This probability should be set according to a priori traffic density estimates based on historical data; higher in areas that are not heavily trafficked and smaller in areas that are not so popular such that, in both cases, it is highly likely that at least one client in the aggregation interval did not upload.

Specifically, we can determine the appropriate probabilities as follows. Let F be a public estimate on a lower bound on the number of cars passing through a statistic point in the corresponding statistics interval. Let P be the desired probability that at least one client does not upload. We need to determine the probability p with which each car should decide to upload.

We have that Pr[at least one does not upload] $= 1 - p^F = P$ and thus,

$$p = \sqrt[F]{1-P} \tag{6.1}$$

The expected number of clients not uploading is equal to $F(1 - \sqrt[F]{1-P})$. We are interested in the probability that a client that traverses L locations where data uploads occur can plausibly deny that it went by those locations. For that to be claimed, each of these locations must have at least one client not uploading data. Hence, the probability that a client can plausibly deny that it traversed path of length L is equal to P^L. Therefore, with an adjustable probability P, at least one client will not upload.

It is important for the aggregator to be uncertain about how many clients passed through a statistics point. If he knew that exactly one person did not upload, he would know that the actual number of tuple through a road is one plus the number of tuples uploaded. Fortunately, if F is the number of people passing through a location, and $N < F$ tuples were uploaded, the probability that N or $N+1$ people upload are very close in value. In particular, the fractional difference between these two probabilities is:

$$\frac{\Pr[N\,\text{upload}] - \Pr[N+1\,\text{upload}]}{\Pr[N\,\text{upload}]} = 1 - \frac{F-N}{N+1}\frac{p}{1-p} \tag{6.2}$$

If we replace N with its expected value Fp, we obtain a fractional difference of $1/(N+1)$, which for popular locations is small. Thus, the probability that $N < F$ people upload is about the same with the probability that $N+1$ people upload. When the aggregator sees

$N < F$ clients, he is uncertain whether there were $N, N-1, N+1$ clients. Alice can now deny plausibly that she went on a path.

We analyze the loss in accuracy of the aggregate statistics resulting from the probabilistic upload in Section 6.6.4; it turns out to be remarkably small.

6.2.4 Strict location privacy (SLP)

We now present a protocol that achieves our strict location privacy definition. We first need to make sure the following requirements hold:

1. Tuples should be uploaded in a way that does not leak when in the statistics interval they were generated.

2. The raw value in the tuple should be encrypted; the aggregator should only have access to the aggregates.

3. The aggregator should not have access to the total number of tuples uploaded for a particular statistic.

The first requirement is needed because if the statistics interval is large enough and the number of clients passing through some locations is sparse, an adversary can trace the paths of some clients using estimates of how long it takes on average to traverse a certain segment. Clients should either send tuples towards the end of the statistics interval or forward them to a proxy or network that will delay them. The need for the second requirement is evident. For example, if a client is a habitual speeder, tracking high speed value uploads will likely indicate his path.

The third requirement is the most challenging. The number of tuples generated leaks privacy. For example, if Alice and Bob are the only people living on a certain street and the aggregator sees two tuples coming from there, likely both Alice and Bob left their house. Therefore, the aggregator needs to receive a number of tuples that is independent of the actual number of tuples generated.

Clients encrypt the values using a homomorphic encryption scheme that allows the aggregator to compute the statistic on encrypted data; the protocol will ensure that the aggregator can decrypt only the result. To obfuscate the actual number of tuples, clients upload some *junk* tuples that are indistinguishable from legitimate tuples and that cancel out during the statistics computation. To ensure that the number of tuples uploaded is independent of the actual tuples generated, clients will upload in total a constant number of tuples for each statistic. Each statistic has a publicly-known standard number of tuples to be uploaded, which is an upper bound on the actual tuples to be uploaded, and which can be computed using the historical popularity of the road.

We discuss the specific encryption schemes we use Section 6.3 together with the statistics they support and how to verify that the auxiliary decrypted the correct value. To cancel out in the aggregations, junk tuples are encryptions of zero or one. Because the encryptions schemes are randomized, they will be indistinguishable from correct tuples. If the value of a junk tuple is used in a sum, it should be an encryption of zero; if it is used in a product, it should be an encryption of one.

To keep the number of tuples uploaded constant, we need an online rendezvous point where clients can somehow synchronize (the clients cannot communicate directly).

Claim 2. *If clients cannot directly communicate and the number of clients uploading at a statistics point is not predictable, they cannot upload a total constant number of tuples in an interval of time at all times.*

The proof is in the appendix. One such rendezvous point is naturally the aggregator. However, the aggregator can count how many clients are requesting the synchronization information and thus estimate how many clients attempt to upload! Having clients make junk requests would require synchronization on the requests to make and this leads to the same recursive problem.

Instead, we augment our protocol with a new online module: an *auxiliary*. However, we assume very little from it: it is *partially trusted* and *has light load*. The auxiliary cannot change the statistics result and cannot collude with clients to let them upload more than their quota. Furthermore, the auxiliary never handles client id and location information. In previous work [35], the intermediaries were heavily trusted; a corrupt intermediary could change the statistics, collude with clients, and had access to complete client information.

The computational burden on the auxiliary is light: it stores the total number of tuples uploaded at the aggregator for a statistics so far, gives this number to inquiring clients, and decrypts one value per interval. We expect in fact that a suitable implementation of the auxiliary would be distributed across public clouds such as on Amazon or Azure, they can be run by the Electronic Frontier Foundation or the government, an accredited company employed by the aggregator, or the participants themselves. If the auxiliary is completely malicious, clients do not have SLP and they will just fall back to PD(which we assume clients always run).

We now specify the protocol for the statistics with id denoted by id and with standard number of tuples to be uploaded S_{id}. The players are: the auxiliary (X), a client (C), and the aggregator (A). At the beginning of the system, X publishes a public key for a homomorphic encryption scheme.

Setup. X records zero for the number of tuples uploaded so far for id, $s_{id} := 0$.

Upload. $C \leftrightarrow X$: Request the number of tuples uploaded so far s_{id} for the statistic of interest.

$C \rightarrow A$: If t is the current time, I is the time interval of id, and S_{id} is the total number to be uploaded for this statistics, then the client should upload $\Delta s := \min(\mathsf{quota}_C, \lfloor S_{id} t/I \rfloor - s_{id})$ tuples, where quota is the maximum number of tuples the client is allowed to upload. One of the tuples is accurate, the others are junk. (In parallel, X computes $s_{id} := \Delta s + s_{id}$ assuming he knows the quota.)

Aggregation. $A \leftrightarrow X$: At the end of the interval, the aggregator computes the aggregate on ciphertext using the homomorphic properties of the encryption. The auxiliary must decrypt only one value per interval and verify the decrypted value.

The reasoning behind the equation for Δs is to keep an approximate value of $S_{id} t/I$ tuples uploaded so far at time t (even though these tuples may be physically uploaded at random times in the interval). We can see that when $t = T$, S_{id} tuples have been uploaded so far.

One recommendation for picking S_{id} is $\mathsf{quota}/2 \cdot F$ (for quota ≥ 3), where F is the historical average of the number of clients for that statistics. We can see that F clients can upload a total number of tuples anywhere in $[0 \ldots F \cdot \mathsf{quota}]$. We can see that this number of uploads can be sustained even by half of the drivers $F/2$ (each uploading at the quota) meaning that even in less trafficked cases privacy is maintained.

The aggregator can also wait for a short period after the statistics were terminated so that few clients traversing the statistics point after the interval is over can complete the uploads for that interval.

Since the aggregator verifies the decryption, the auxiliary cannot change the result. The auxiliary can be replicated at different service providers or cloud computing infrastructures, or in a peer-to-peer system in case some of them refuse to provide an answer, and be partitioned to scale.

Note that encryption of values permits malicious clients to attempt to upload exorbitant values to bias the statistics. This can be addressed with the following refinement of the protocol: Depending on the particular statistic, the aggregator publishes an interval within which reasonable values must live. The client then additionally uploads a zero-knowledge proof that the value in the encryption lies in the interval (using the efficient scheme of [10] which we will evaluate in the context of the accountability protocol).

Theorem 3. *If the number of uploads for each statistics is the desired standard value, the encryption scheme is semantic secure, the auxiliary does not leak upload times and only decrypts the overall result, no privacy is leaked according to Definition 5.*

Please find the proof in the appendix.

Spatially distributed fake tuples. In order to handle situations in which streets of interest for the aggregate statistics computation have low densities, one might use the following extension of the SLP protocol: whenever a client decides to upload, the client uniformly chooses k statistics with the given time interval (i.e., k different locations) and engages in the SLP protocol with the auxiliary and aggregator for each of these statistics — however, all tuples are fake except for the tuple associated to the real upload. For sufficiently large k, all of the statistics will have S_{id} uploads. Note that this protocol requires enlarged quotas, and thus malevolent clients have greater latitude to affect the results.

Discussion. We considered a variety of solutions which remove the need for the auxiliary by placing its functionality securely on the aggregator. Given the restrictions on a practical system outlined in Section 4.2, we do not think this is possible.

For instance, it would be attractive to use a solution in which the aggregator can decrypt the aggregate value itself, in a way that it can only decrypt once. However, this seems to require "distributed threshold cryptography" which is impractical given the no-coordination assumptions we are working with. Regarding the second task of the auxiliary, informing clients of the number of tuples presently uploaded, plausible solutions involved private information retrieval that would hide the statID requested by the client to the aggregator; these solutions, however, are highly impractical to perform at high speeds and for large amounts of data. Even if private information retrieval is as practical as possible, an inherent limitation is that the aggregator has to scan through all possible statistics id per request.

Furthermore, we investigated "statistical obfuscation" when computing averages, an intuitive possibility for allowing the server to compute an overall result yet not learn about individual values. Unfortunately, it is hard to provide formal guarantees about the indistinguishability of junk tuples and real tuples with such a solution: for a significant proportion of tuples, the noise added to real tuples will result in values that could not have been created by junk tuples.

6.3 Statistics Supported

PD supports essentially *any* aggregate statistics computation because the data is uploaded in plain text; the aggregator can use it in any computation of his choice. SLP is also quite general. There are practical homomorphic encryption schemes for three basic computations:

1. *Summations (S).* Using EC-ElGamal, Paillier, or Benaloh schemes [6], the aggregator can aggregate an arbitrary number of ciphertexts and obtain a ciphertext for the summation of the underlying plaintexts.

2. *Many summations, one multiplication, many summations (SMS).* Using the Boneh-Goh-Nissim cryptosystem [9], the aggregator can perform an arbitrary number of summations on encrypted values, followed by one multiplication, and finally followed by any number of additions.

3. *Multiplications (M).* Using RSA, ElGamal, or SMS, the aggregator can aggregate an arbitrary number of ciphertexts and obtain a ciphertext for the product of the underlying plaintexts.

Thus, SLP supports *any aggregation that is reducible to the three operations.* Fortunately, these operations are fairly general and support a wide range of statistical functions. Table 6.1 lists some of the ones supported by PrivStats, including many of practical interest.

S, M, and SMS do not support median, min, and max. As such, we propose some partial remedies for these situations. To compute the median, one can estimate it with the mean in some cases. To compute min and max, clients need to upload an additional bit with their data. The bit is 1 if the statistic value is smaller than a certain threshold. The aggregator can collect together all values with the bit set and ask the auxiliary to identify and decrypt the minimum.

The schemes we use have the nice property that the aggregator can verify the decrypted value from the auxiliary. For the Boneh-Goh-Nissim cryptosystem (to which the other schemes reduce), an encryption to x is in fact a commitment to x, so the auxiliary can provide a decommitment key as a proof of correctness as in [53].

6.4 Accountability

Since each client uploads statistics anonymously, some malicious clients may attempt to bias the statistics by uploading multiple times. In this section, we discuss PrivStats's accountability protocol, which enforces quotas on how much each client can upload.

There is tension between maintaining the location privacy of each client and placing a quota on how much data they can upload. The idea is to have each client upload a cryptographic token whose legitimacy the aggregator can check and which does not reveal

Table 6.1: Table of statistics supported by PrivStats with example applications and implementation. Conditional aggregations are aggregations computed only over statistics that satisfy a certain predicate. For example, we may want to know the average speed of individuals whose speeds fell inside a range (or were greater than a threshold).

Aggregation	Example Application	Implementation
Count	Traffic congestion: no. of people through an intersection.	Each driver uploads encryption of one when they pass by the statistics point. Aggregator uses S.
Summation	No. of people carpooled through an intersection.	Each car uploads encryption of the number of people in the car. Aggregator uses S.
Average	Average speed, delay estimation.	Clients upload an encryption of their speed/delay and an encryption of the value one (for count). Junk tuples will correspond to two encryptions of zero. Aggregator uses SMS by adding speeds, counts and dividing them.
Std. Dev.	Studying std. dev. of delays.	Compute average and count as above. Compute sum of square values using SMS.
Product	Existence: Is there at least one client satisfying a predicate?	If a client satisfies the predicate, he uploads encryption of zero, else he encrypts a random number (similarly for junk tuples). Using M, one client uploading zero will induce zero in the result. M can also be used for parity checks or summing small values using the exponential.
Conditional aggregations of all the aggregations above	Count of people exceeding speed limit	Clients upload encryption of the neutral element (zero or one) if the condition in question is not satisfied. Besides this change, the clients and the aggregator follow the protocol for the desired aggregation above.
Any other aggregation reducible to S, M, SMS.		

any information about the client; yet a client cannot create more that its quota of legitimate tokens for the same statistic and location.

Notation and conventions. Let SK_A be the aggregator's secret signing key and PK_A the corresponding public verification key. Consider the statistics with identifier id. Let T_{id} be a token uploaded by a client for statistic id. All the values in this protocol are computed in $\mathbb{Z}_n$, where $n = p_1 p_2$, two safe prime numbers.

We make standard cryptographic assumptions that have been used in the literature (e.g., the strong-RSA assumption). Further specific assumptions and more background is discussed in the Appendix.

6.4.1 Protocol

For clarity, we explain the protocol for a quota of one tuple per statistics id per client and explain how to use the protocol for a greater quota.

We have the following goals:

1. *Accountability:* The aggregator should examine T_{id} and realize if the driver has uploaded a tuple before for id.

2. *Location privacy:* The aggregator should not be able to link different tokens from the same client because it will learn path information.

We solve this problem by designing an efficient *zero-knowledge proof of knowledge* protocol from scratch. Proofs of knowledge [30], [5], [60] are proofs by which a machine can prove that it knows some value that satisfies a certain relation. For example, Schnorr [60] provided a simple and efficient algorithm for proving possession of a discrete log. If we add the zero-knowledge [30] property, we also have the guarantee that no information about the value in question is leaked.

Registration. Before a client C can participate in aggregations, he needs to register with the aggregator A. C identifies himself to A (using his real identity) and A will give C one capability (allowing him to upload one tuple per statistic) as follows. C picks a random number s and obtains a blind signature from A on s (using the scheme in [14], which is accompanied by a proof of knowledge of a signature for a value that is hidden inside the given commitment).

Let $\mathsf{sig}(s)$ denote a signature from the aggregator on s. s is used to create a token T_{id} for every id. The client keeps s and $\mathsf{sig}(s)$ secret. Since the signature is blind, A never sees s or the signature. If the signature were not blind, the aggregator could a client's real identity to a capability; he could then test each token received to see if it was produced by the capability of the client in question. By running the signing protocol once for a client C, A can ensure that he gives only one capability to C. The signature attests that the capability of the client is correct. Without some certification, a client can create his own capabilities and upload more than he is allowed to.

Accountability during upload

1. $C \rightarrow A$: Computes $T_{\mathsf{id}} = \mathsf{id}^s \mod n$ and uploads it together with the data. C also uploads a zero-knowledge proof of knowledge (as explained below) that he knows a value s such that $\mathsf{id}^s = T \mod n$ and for which it has a signature from the aggregator.

2. $A \rightarrow C$: Aggregator checks the proof and if the value T has been uploaded before for id. If the latter holds, it means that this tuple is an over-upload and discards it.

Intuitively, T_{id} does not leak any information about s because of hardness of the discrete log problem. The client's proof is a zero-knowledge proof of knowledge (ZKPoK) so it does not leak any information about s either. Since T_{id} is computed deterministically, more than one upload for the same id will result in the same value of T_{id}. The aggregator can detect these over-uploads and throw them away. The client cannot produce a different T_{id} for the same id because he cannot convince the aggregator that he has a signature for the exponent of id in T_{id}. Here we assumed that id is randomly distributed (which can be enforced by hashing id in practice).

To enforce a $\mathsf{quota} > 1$, the aggregator simply issues the client more capabilities (quota of them) during registration.

6.4.2 ZKPoK

We now describe the zero-knowledge proof of knowledge, which we constructed from scratch. We require a commitment scheme (such as the one in [53]): recall that this allows Alice to commit to a value x by computing a ciphertext C and giving it to Bob. Bob cannot learn x from C. Alice can open the commitment by providing x and a decommitment key that are checked by Bob. Alice cannot open the commitment for $x' \neq x$ and pass Bob's verification check.

Public inputs: id, T_{id}, PK_A, g, h
Client's input: s, $\sigma = \mathsf{sig}(s)$.
Aggregator's input: SK_A

Construction 1. ***Proof that*** C ***knows*** s ***and*** σ ***such that*** $\mathsf{id}^s = T$ ***and*** σ ***is a signature by*** A ***on*** s***.***

1. *C computes $\mathsf{com} = g^s h^r \mod n$, where r is random. com is a Pedersen commitment to s. C proves to A that he knows a signature σ from the aggregator on the value committed in com using the protocol in [14].*

2. *C proves that he knows s and r such that $T_{\mathsf{id}} = \mathsf{id}^s$ and $\mathsf{com} = g^s h^r$ as follows:*

   (a) *C picks k_1 and k_2 at random in $\mathbb{Z}_n$. He computes $T_1 = g^{k_1} \mod n$, $T_2 = h^{k_2} \mod n$ and $T_3 = \mathsf{id}^{k_1} \mod n$ and gives them to the aggregator.*

   (b) *The aggregator picks c a prime number, at random and sends it to the client.*

   (c) *The client computes $r_1 = k_1 + sc$, $r_2 = k_2 + rc$ and sends them to the aggregator.*

   (d) *The aggregator checks if $\mathsf{com}^c T_1 T_2 \overset{?}{\equiv} g^{r_1} h^{r_2} \mod n$ and $T^c T_3 \overset{?}{\equiv} \mathsf{id}^{r_1} \mod n$. If the check succeeds, it outputs "ACCEPT"; else, it outputs "REJECT".*

This proof can be made non-interactive using the Fiat-Shamir heuristic [22] or following the proofs in [10].

Theorem 4. *Under the strong RSA-assumption, Construction 1 is a zero-knowledge proof of knowledge that the client knows s and σ such that $\mathsf{id}^s = T_{\mathsf{id}} \mod n$ and σ is a signature by the aggregator on s.*

The proof is presented in the Appendix.

6.5 Implementation

We implemented the SLP protocol and accountability protocol in C++. The implementation consists of three modules: the *Client,* who produces tokens according to the accountability protocol and uploads encrypted junk/correct tuples, the *Aggregator,* who checks

Table 6.2: Running time of various operations.

Party	Setup (ms)	Registration (ms)	Upload (ms)
Client	0	70	320
Aggregator	160	70	310
Total	160	140	630

tokens according to the accountability protocol, collect tuples per statID, and at the end compute the result using homomorphic encryption and ask the auxiliary for the result, and the *Auxiliary,* who decrypts the result from the aggregator.

The accountability protocol implementation includes the registration and upload protocol for both the client and the aggregator as described in Section 6.4. The upload protocol includes our zero-knowledge proof of knowledge from Section 6.4; we also implemented the signature scheme by [14].

Our implementation is only 500 lines of code for all three parties, not counting empty lines. This count does not include basic libraries and NTL (the number theory library).

6.6 Evaluation and Analysis

We ran our experiments on a low-end desktop PC—a dual-core processor with 2.2 GHz and 1 GByte of RAM. Except for the registration phase, we expect the clients will run on more limited hardware such as a smart phone. We emulated the environments for two smartphones, Nexus and iPhone, as described below.

6.6.1 Run time

We first evaluate the performance of the accountability protocol alone and then evaluate each party's performance as a whole. The homomorphic encryption scheme we used in our evaluation is Benaloh [6], which allows summations and counts. In our implementation, encryption takes 0.05 ms and decryption takes 76 ms for an integer. The decryption time is much slower because it involves searching the plaintext in a small field of possible values. The ciphertext length is 1024 bits.

Accountability protocol. The client always needs to compute and upload tokens, but the server can choose to do so probabilistically. Table 6.2 shows the code execution time of this protocol (we analyze network overhead later).

The upload protocol requires 320 ms of processing on the client side and 310 ms on the aggregator side. Note that the overall protocol takes three transmissions: the client computes and uploads some data (phase 1), the aggregator answers with some random numbers used for the challenge (phase 2), the client responds with verification data (phase 3), and the aggregator checks the verification results (phase 4). We measured the execution time for each phase, and the time of phase 2 and phase 3 are essentially 0 milliseconds. The reason for this is that in phase 2, the aggregator just picks a few random numbers and sends them to the client, and in phase 3 the client performs a few multiplications and additions. As optimizations, clients can preprocess tokens or postprocess tokens and upload tuples at a later time; the aggregator can also probabilistically check tokens

Table 6.3: Client execution time for one tuple generation and upload for different execution frameworks.

Machine	Hardware capabilities	Upload
Nexus	512MB RAM, 1GHz	352ms
iPhone	256MB RAM, 600MHz	660ms
PC	1GB RAM, 2.2GHz	320ms

Client. The client runs two phases: registration and upload. Registration consists entirely of the accountability protocol evaluated above and can be run on a commodity PC. Upload consists of generating a tuple and uploading it; the performance is dominated by the accountability and encryption times above. Since the client will run on some more limited hardware (such as a smartphone), it is important to understand the performance of our protocols on such devices. To that end, we emulated the computational capabilities of the Nexus and iPhone platforms using a Linux virtual machine. We adjusted the resources of a virtual machine to match the RAM sizes of these two smartphones.

Unfortunately, one cannot adjust the length of a CPU cycle, so we evaluated the runtime in terms of clock cycles rather than actual time and then scaled the clock cycle to the device considered. If the same CPU-intensive binary can run on two different machines of different clock cycles (and all other parameters are equal) the scaling should be approximately proportional. The results we obtain are not exact because the operating systems are different, but we believe that this approach gives a very good idea of the performance on these devices. We are currently working on porting our system and needed libraries to Java to run on Android.

Table 6.3 shows our performance results. On the Nexus, which is quite fast, the results are about the same as on the PC. On the iPhone are slower, but an overhead of 0.6 seconds per upload is still reasonable given that a client uploads a tuple at a much lower time period than that.

Every upload incurs an additional network round trip delay because the client contacts the auxiliary. We believe this is reasonable. If the client has intermittent connectivity, it can contact the auxiliary at a time t^* before the time t at which it actually uploads. It can then estimate the number of tuples uploaded so far, s_t for the statistic of interest using $s_{t^*}t/t^*$ and operate with that estimate.

Aggregator. The aggregator's role consists of two phases: collect tuples and compute statistics. Collecting the tuples involves verifying the token for correctness and uniqueness. Again, verifying the correctness of the token (accountability) dominates execution time. By turning off this verification, the aggregator could process 10^6 tuples in 8s (mostly uniqueness checks).

Typically, aggregation on encrypted data corresponds to multiplying all the data items together. The aggregator can perform this task as it receives the tuples. For illustration purposes, the aggregator can aggregate (in one computation) 10^5 values of 1024 bits in size in 0.65 s, $5 \cdot 10^5$ values in 3.28 s and 10^6 values in 6.46 s; the scaling is linear.

We would like to compute how many statistics an aggregator can support. It takes 310 ms to serve the upload of one client if the aggregator verifies the accountability proof. Let p be the probability with which the aggregator checks a certain proof. Let n be the number

Table 6.4: The amount of data sent out by each party during various protocols.

Party	Registration	Upload	Aggregation
Client	1KB	7.01KB	0
Aggregator	0.5KB	0.5KB	128B
Auxiliary	0	2B	4B
Total	1.5KB	7.51KB	132B

of drivers uploading for a statistics. Therefore, one aggregator can support $3 * 10^5/np$ statistics in one day. For instance, statistics [50] show that there are about 2000 cars on average passing in an hour through a highway lane so $r = 0.556$ so let $n = 2000$. If $p = 1$, this amounts to ≈ 150 and if $p = 0.1$, it amounts to 1500 popular statistics in a day. Therefore, we expect one commodity aggregator node to suffice for computing the statistics in a city.

Auxiliary. The auxiliary's processing is very light, reducing in effect to one decryption (76 ms) of the statistics result per statistics computed. The ratio of the auxiliary's processing time to the aggregator's is about $76/(310 \cdot N)$, which is $\approx 10^{-4}$ for $N = 2000$. Of course, there is still the overhead of responding to requests to clients involving the number of tuples uploaded for a statistics, but this should be similar (if not smaller) than the overhead the aggregator incurs when receiving tuples from clients.

We measure the bandwidth and storage overhead in the appendix. Both are quite small: about 7 Kbytes per token including the proof of knowledge, and 137 bytes for the other fields (statistic value, pointstamp).

6.6.2 Bandwidth and storage overhead

We measure the amount of data sent by each party during our SLP protocol; Table 6.4 presents our results.

The size of a tuple consists of 6.875 Kbytes for the token (with the proof of knowledge included) and 137 bytes for the other fields (statistics value, pointstamp).

The client should not maintain state unless he wants to preprocess some tokens. The aggregator needs to maintain, for each statistic, 8 bytes of data corresponding to a statistic id and a number of tuples uploaded so far.

The aggregator should maintain all the tokens (1024 bits each, without the proofs of knowledge) to make sure that the tokens uploaded are unique, a requirement for the accountability protocol. With a RAM of 1GB, an auxiliary can store in main memory data for 125,000 statistics. Thus, storage should not be a problem.

6.6.3 Accuracy of the statistics

As mentioned, we prevent clients from uploading more than a small constant number of tuples for a statistics ID and we enforce that the value uploaded be in a certain interval of possible values; however, we of course cannot verify the value the client uploads, other than ensuring that it lies in an interval of acceptable values.

If a client wants to bias the statistics in a certain direction (e.g., show that the average speed on a road is very small), in the worst case, he will upload the minimum or maximum acceptable value. We analyze how much a client can affect the statistics. Let N be the number of clients uploading for a certain statistics, I is the interval of accepted values, and μ is the average speed if all clients are honest. For example, using the statistics in [50], N can be 2000 clients in an hour. For average speed computation, $I = (30, 100)$ mph. μ might be 60 mph, for instance. The highest change in the average that a client can induce is $\pm|I|/N$, where $|I|$ is the length of the interval I; this happens when the average is in fact the minimum or the maximum value in the interval and the client uploads the maximum or minimum, respectively). The maximum fractional error is $\pm|I|/N\mu$ or $\pm|I|/S$, where S is the sum of all values uploaded by the clients. A client can affect a statistics by ratio of the size of the interval of acceptable values and the sum of all values uploaded by other clients. The error increases linearly in the number of malicious clients and is inversely proportional to the total number of clients. We can see that if $|I|$ is not very large and N is large (a popular area), the error introduced by a client is small. For our example, this is equal to ± 0.035 mph and 0.06%, both rather small.

6.6.4 Plausible deniability

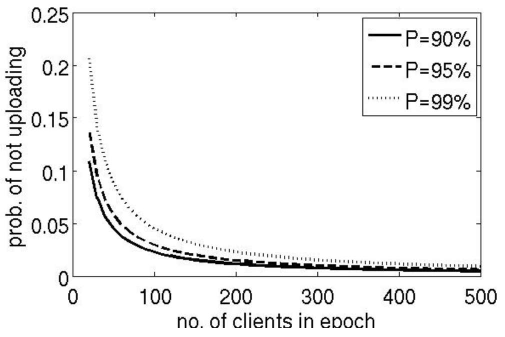
In this section, we show that guaranteeing plausible deniability in the presence of side information even for the entire population does not affect the number of tuples uploaded significantly (and hence does not affect the accuracy of the aggregate result).

Figure 6-2(a) illustrates the probability that a car does not upload and Figure 6-2(b) shows the expected number of cars that do not upload given the popularity of certain pointstamps. We can see that this probability is rather low and decreases. It is interesting to note that the number of clients who do not upload during each interval at each location is *constant*.

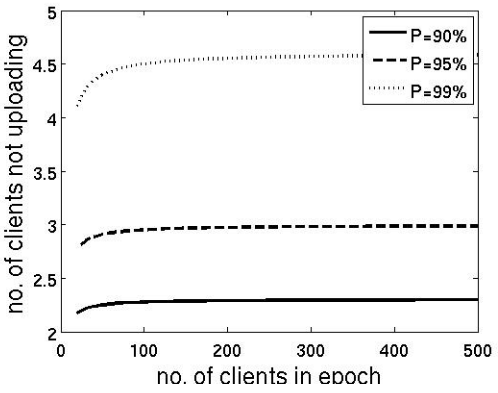
The reason is that $\lim_{F\to\infty} F(1-\sqrt[F]{1-P}) = -\ln(1-P)$. Moreover, as we can see from the graphs, the convergence is rapid. The expected number of clients not uploading is a small constant (e.g. 3 for $P = 0.95$) approximately *independent* of the number of clients traversing a location. This means that accuracy of the statistic is not affected and the percentage of clients not uploading decreases with popularity in an inversely proportional manner.

To understand how much impact the tuples not uploaded can have on the overall statistics, we examine real data collected in CarTel [38]. We obtained real traces from the CarTel project testbed, containing the paths of 27 taxis mostly in the Boston/MA area over one year. Each car drives many hours every day. The cars carry GPS sensors that record location and time, which we match against a map database. The traces consist of tuples of the form (car tag, segment tag, time) generated at intervals with a mean of 20 seconds.

Because 27 drivers cannot give a good sense of what intersections are popular, we look at each one-day path of a driver as an independent path; there are about 4800 one-day paths in a month. We chose varying lengths for the intervals of the statistics. For each interval, we computed how many one-day paths are on each road segment in the same time interval. We consider that a statistic is recorded only at points with at least 10 cars passing through in the whole aggregation interval. We computed the average ratio of clients who do not



(a) Probability that a client will not upload as depending on the popularity of a certain road for different plausible deniability probabilities P.



(b) The expected number of clients that do not upload.

Figure 6-2: PrivStats' analysis results.

upload to the total number of clients passing through some intersection over all intersections (as a percentage); the result is in Figure 6-3. The percentages are small, which indicates that our scheme for plausible deniability does not significantly affect the number of tuples to be uploaded.

If a certain probability $\mathbb{P}$ of having plausible deniability on a road of length L is desired, we need to pick $P = \mathbb{P}^{1/L}$. For example, to have plausible deniability with 0.95 on an entire path that passes through 5 statistic points, we need to use $P = 0.99$.

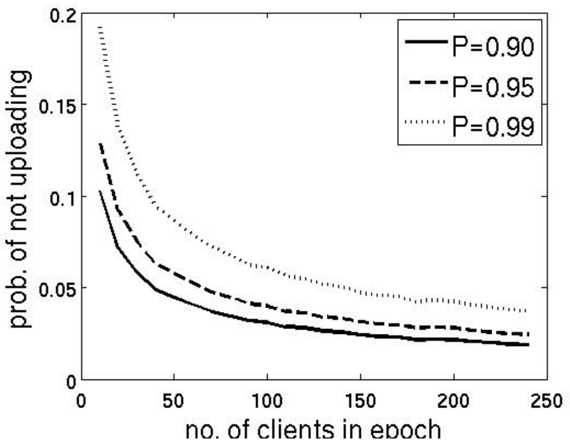


Figure 6-3: Fraction of users not uploading data for various lengths of the statistics interval for CarTel data in the year 2008.

Chapter 7

Other Applications

In this section, we suggest applications of VPriv and PrivStats to location-based mobile systems that are not vehicular. Both VPriv and PrivStats can support any application that complies with their model (Chapter 4). In what follows, we explain how the model of VPriv and PrivStats applies to other applications.

7.1 Location-based social networking applications

In recent years, with the increased popularity of smartphones, a variety of location-based social networking sites have been developed [59]. Currently, most such applications violate location privacy: the client application uploads the client's identifier together with location data to the application's server. Besides being a breach of privacy, location disclosure could also lead to serious safety consequences [54]: some users were robbed after indicating on a social application that they were away from home (e.g., on vacation); some women were stalked and threatened because the application provider negligently made available certain location information users considered private. Moreover, some location-based social applications (e.g. Foursquare, Gowalla) post on their public website feeds of location updates from users.

Moreover, in some of these social applications, the server just needs to compute some functions of the users' paths and does not need to know the actual path. VPriv and PrivStats could be employed in such cases and we describe a few examples below.

One area of applicability of VPriv are social games such as Foursquare [23] and Gowalla. In Foursquare, users earn points based on where, when, and how often they check in with Foursquare. In addition, if users check in at a certain location (e.g., restaurant) more than a threshold number of times, they earn points and may get rewards from the owner of the place. Using VPriv, a server could compute all these functions without knowing the path of the users. Users could upload similar time-location tuples, where the location is replaced with the name of a place (e.g., a restaurant's name). The server could assign points to each time-location tuple received and compute the total number of points of a user using VPriv.

PrivStats could be applied to social applications that compute aggregate statistics over locations, such as Loopt [20] for restaurant ratings. In Loopt, customers upload ratings of each place and Loopt (in partnership with Yelp) maintains the average rating. PrivStats could be used to maintain the anonymity of users uploading, enforce quota on how much each uploads, as well as compute the statistics at the server.

Of course, there are location-based social networking applications where neither VPriv nor PrivStats are directly applicable. An example problem is, given a certain user and its location, to find which friends of the user are within a certain perimeter. Approaches in which the server runs a secure multi-party computation protocol for each friend of a user to determine which friends are within the desired perimeter from the user seem more appropriate [64].

7.2 Applications to cellular networking

An interesting application for VPriv (and for PrivStats, but to a lesser extent) is cellular phone networking. Every time a user calls another user or is being called, the user communicates with the tower of control. As the user moves, the user may change towers of control. This may especially happen when driving or traveling. All throughout, the towers of control with which the user communicates report to the cell phone company's logical server; the server can use the location of the towers of control and create an approximate path of the users. The cell phone company can leak the path due to software bugs or hacker attacks, or simply release it upon a subpoena. Obviously, one of the utilities of maintaining records for each user is to compute the bill a user has to pay at the end of the month.

In principle, VPriv could be used to allow the cell phone company to charge users, yet not know their paths. In what follows, we will outline, at a high level, how VPriv would work in this setting. Of course, there are many other details that one needs to work out when performing an actual implementation (and potentially political barriers to overcome), but here we just want to propose and sketch such an idea. An interesting future project would be to work out the details of such an implementation.

There is a delicate aspect we need to be careful with: the server needs to know where a user (say Alice) is, not only for purposes of billing, but also to allow Alice to receive or issue calls. We must explain how phone calls happen if VPriv were used. Consider the random tags that users upload together with their tuples in VPriv. In the phone setting, each user generates the random tags using a pseudorandom function (see Section 3.2.2) that is indexed by a secret value each users has. This secret value is unknown to the server. We can think of this secret value as the base phone number of the user. This base phone number will be used to generate other phone numbers that will actually be used in communication. The time is divided in short time intervals. Each random tag is generated by applying the pseudorandom function to the identifier of the time interval. The time partitioning algorithm into time intervals is known to everyone.

$$\text{random tag for time interval } I = f_{\text{base phone number}}(I)$$

Intuitively, each random tag becomes a temporary phone number to be used only in the corresponding time interval. If these random tags are computed in a very large field, collisions become unlikely. Therefore, the base phone number is used to generate a lot of temporary pseudorandom phone numbers. Each user keeps their phone number secret and only gives it to people who are allowed to call the user. The user follows the same procedure for the random tags as in VPriv to register with the server. Also, similarly to VPriv, the server does not know to what user a temporary phone number belongs.

As Alice moves around, she checks in with the tower of control and informs it of the temporary phone number corresponding to that time interval. This allows the tower of control to know where the user with the temporary phone number is located when someone attempts to call that phone number. If Bob wants to call Alice and he has Alice's base phone number, he can use the pseudorandom function and compute which is Alice's temporary phone number at the time of a call. Then he can issue the call and the server will be able to route it to the appropriate tower of control.

If a friend of Alice is malicious and posts Alice's base phone number online, everyone can construct all future temporary phone numbers and thus be able to call Alice. Even without VPriv, if Alice's phone number becomes public, everyone will be able to call her until she changes her phone number. However, we need to prevent users from advertising one of Alice's temporary phone numbers to the tower of control and making Alice pay for charges. For this, Alice needs to obtain a blind signature from the server on each temporary phone number during registration (see Section 3.2.4 for a definition of blind signatures). Whenever Alice advertises a temporary phone number to the tower of control, she must provide the blind signature. Users other than Alice could not have obtained such a signature.

Like in VPriv, the server should maintain mappings of random tags (or temporary phone numbers) to the cost of the conversation. During the reconciliation phase (at the end of the month), the server and the client compute the bill of the client following the protocol in VPriv 5.2.

Chapter 8

Conclusions

In this thesis, we developed two practical systems, VPriv and PrivStats, for protecting location privacy of users while efficiently supporting a wide range of location-based vehicular services as well as other mobile applications.

We formalized various definitions of location privacy and introduced novel and practical cryptographic protocols (function computation in VPriv, accountability and statistics computation in PrivStats) that provably achieve such definitions. We implemented both systems and verified that they are efficient enough to run on smartphone-class devices or commodity computers.

We hope that VPriv and PrivStats will be used to reduce privacy concerns associated with location-based vehicular and mobile services, and thus allow such systems to reach their full benefits.

Appendix

Proofs

Proof of Theorem 1. 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. After the client shuffles at random the pairs (s_j, t_j), the server cannot tell which $f_k(s_j)$ corresponds to which s_j. Without such shuffling, even if the s_j is encrypted, the server would still know that the j-th ciphertext corresponds to the j-th plaintext. This will break privacy in Step (iv) for $b = 1$ when the server compares the ciphertext of s_j to the ciphertext of v_j.

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. Thus, 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, the server cannot find v_i. Providing D only provides decommitment to the sum of the tolls which is the result of the function, and no additional information is leaked (i.e., 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. □

Proof of Claim 2. Consider two alternate scenarios in which N_1 and N_2 ($N_1 \neq N_2$) clients upload for the same statistics point. Assume that in the first case client i uploads $n_i^{(1)}$ tuples and in the second case, he uploads $n_i^{(2)}$ tuples. There must be at least one client i for which $n_i^{(1)} \neq n_i^{(2)}$ because otherwise the total numbers of tuples uploaded in the two scenarios are different (because $N_1 \neq N_2$). Since the number of clients for a statistics is unpredictable and clients cannot communicate, client i cannot know in which scenario he is. Since both scenarios have nonzero probability of happening, client i must upload $n_i^{(1)}$ at some point and with nonzero probability; however, when he uploads $n_i^{(1)}$ it can happen that we are in scenario and all other clients uploaded their values for scenario two. In this case, the total number of tuples uploaded is smaller than standard. □

Proof of Theorem 3. According to definition 5, we need to argue that any information that reaches the aggregator (besides the result of the aggregation and any side information he has) cannot possibly leak any additional information about the paths of the drivers. The information that reaches the aggregator consists of a number of encrypted tuples for each statID, each tuple arriving at a certain time in the interval. As we have discussed, the tuples should all be delayed through a Tor or proxy to reach the aggregator towards the end of the statistics time interval. Moreover, the number of tuples that arrive is known to the aggregator already and discloses no information beyond what the aggregator already had. The values of the tuples are indistinguishable from random numbers according to semantic security of the encryption.

We use a simulator argument (which forms the foundation of zero-knowledge proofs) to argue that *the aggregator does not learn any information from the tuples that he did not have already.* Even before the system was running (i.e., when there are no members in the system), the aggregator could have generated S_{id} random numbers for each statistic id and have them all sent to himself at the end of the statistics interval. The distribution of these tuples is the same with the distribution of tuples received from the actual system.

Since the aggregator does not obtain any information from the tuples, the probability of guessing the path of a client is the same as without the data from the clients. □

To prove Theorem 4, we use standard zero knowledge proofs of knowledge definitions (specifically the ones in [46]). For our protocol, the Prover is the client and the Verifier is the aggregator. The proof of the theorem uses the following lemma.

Lemma 5. *Under the strong RSA assumption, Step 2 of Construction 1 is an honest-verifier zero-knowledge argument system with proof of knowledge property for the desired problem.*

Proof. We need to prove the three properties of zero-knowledge proofs of knowledge presented in Section 3.3.2.

Completeness: If the client knows s and r, the aggregator will accept. Let us verify that the checks in Step 2d succeed for an honest client. We have $\mathsf{com}^c T_1 T_2 = (g^s h^r)^c g^{k_1} h^{k_2} = g^{k_1+sc} h^{k_2+rc} = g^{r_1} h^{r_2}$ and $T^c T_3 = (\mathsf{id}^s)^c \mathsf{id}^{k_1} = \mathsf{id}^{k_1+sc} = \mathsf{id}^{r_1}$. Thus, for an honest client, the aggregator will accept.

Proof of knowledge: We must provide an efficient knowledge extractor, K. Consider a prover P that makes the verifier accept with a nonnegligible probability. Let K be a machine that performs the following:

1. Start a random instance of the Prover. Let it output T_1 and T_2.

2. Pick c_1 at random and send it to the prover. Receive r_{11} and r_{12} and see if checks in Step 2d verify. If the checks do not verify, start this algorithm from the beginning.

3. If the checks verify, rewind the Prover up to the point where he just output T_1 and T_2 (the same as above). Pick c_2 at random and send it to the Prover. Receive r_{21} and r_{22}. If the checks in Step 2d are not verified, start this algorithm from the beginning.

4. If $c_1 - c_2$ does not divide both $r_{11} - r_{21}$ and $r_{12} - r_{11}$, start this algorithm from the beginning.

5. If the checks verify, then output $s = \frac{r_{11}-r_{21}}{c_1-c_2}$ and $r = \frac{r_{12}-r_{22}}{c_1-c_2}$

Since the Prover has non-negligible probability of convincing the Verifier, we can see that the equality checks will succeed and we will execute Step 5 in expected polynomial time. [16] prove that Step 4 will happen in expected polynomial time, if the strong RSA assumption holds. Let us show that, in this case, s and r are indeed solutions to our problem.

Since the checks verify, it means that we have $\mathsf{com}^{c_1} T_1 T_2 = g^{r_{11}} h^{r_{12}}$, $T_{\mathsf{id}}^{c_1} T_3 = \mathsf{id}^{r_{11}}$, $\mathsf{com}^{c_2} T_1 T_2 = g^{r_{21}} h^{r_{22}}$, and $T_{\mathsf{id}}^{c_2} T_3 = \mathsf{id}^{r_{21}}$. By dividing the appropriate two equations, we can compute: $g^s h^r = g^{\frac{r_{11}-r_{21}}{c_1-c_2}} h^{\frac{r_{12}-r_{22}}{c_1-c)2}} = (g^{r_{11}-r_{21}} h^{r_{12}-r_{22}})^{1/(c_1-c_2)} = (\mathsf{com}^{c_1-c_2})^{1/(c_1-c_2)} = \mathsf{com}$ and $\mathsf{id}^s = \left(\mathsf{id}^{r_{11}-r_{21}}\right)^{1/(c_1-c_2)} = \left(T_{\mathsf{id}}^{c_1-c_2}\right)^{1/(c_1-c_2)} = T_{\mathsf{id}}$

Zero-knowledge: We must provide a simulator proof. We follow the proof in [56], which proves that the Schnorr [60] algorithm for proving knowledge of a discrete logarithm is zero-knowledge.

We construct a simulator that can generate a communication transcript with any verifier indistinguishable from a transcript between the prover and the verifier. The simulator performs the following. Picks c', k_1', r_1' and r_2' at random. Computes $T_1' = g^{k_1'}$, $T_2' \equiv g^{r_1'} h^{r_2'} \mathsf{com}^{-c'} g^{-k_1'} (\mod n)$ and $T_3' = \mathsf{id}^{r_1'} T_{\mathsf{id}}^{-c'}$. Sends T_1', T_2', T_3' to the verifier. Receives c from the verifier. If $c \neq c'$, restart the verifier and repeat this algorithm from the beginning. If $c = c'$, send r_1' and r_2' to the verifier; output the transcript produced in this run.

Let us argue that this transcript $(T'_1, T'_2, T'_3, c', r'_1, r'_2)$ is indistinguishable from a transcript between the prover and the verifier. We drew c', k'_1, r'_1 and r'_2 at random and computed T'_1, T'_2, T'_3 based on these values. The distribution of these values is indistinguishable from choosing c', k'_1, T'_2, T'_3 at random and then considering random values of r'_1 and r'_2 that satisfy the verifier checks. The latter distribution is the same with the distribution in a real transcript.

This simulation has an expected polynomial time of success if the space of possible values for c is constrained to a polynomial-sized space. □

Proof of Theorem 4. Camenish and Lysyanskaya [14] show that the protocol in Step 1 is a zero-knowledge proof of knowledge. Lemma 5 shows that Step 2 is also a zero-knowledge proof of knowledge.

All we have to prove is that the client cannot run these two proving protocols for different values of s. We proceed by contradiction. Assume that the user proofs knowledge of s_1 according to the proof in Step 1 and knowledge of s_2 according to the proof in Step 2 such that $s_1 \neq s_2$. Let M be a machine that has access to the knowledge extractor for the first and second proofs. M can use these two extractors and obtain $s_1 \neq s_2$ and r_1, r_2 such that $C \equiv g^{r_1} h^{s_1} \equiv g^{r_2} h^{s_2} (\mod n)$. This means that $g^{r_1 - r_2} \equiv h^{s_1 - s_2} (\mod n)$. M can repeat this experiment for different randomness given to the extractors and obtain $s_1 - s_2$ which divides $r_1 - r_2$ in expected polynomial time. By dividing these values, M obtains the discrete log of h in base g and thus invert the discrete log problem. □

Statement of Originality

The results contained in this thesis are original research that is joint work with Hari Balakrishnan and Andrew J. Blumberg. They are contained in the following papers:

- Raluca Ada Popa, Hari Balakrishnan, and Andrew J. Blumberg. VPriv: Protecting Privacy in Location-Based Vehicular Services. USENIX Security, 2009.
- Raluca Ada Popa, Andrew J. Blumberg, and Hari Balakrishnan. PrivStats: Location Privacy and Accountability in Aggregate Statistics for Mobile Systems. In submission.

Bibliography

[1] E-ZPass Records out Cheaters in Divorce Court.

[2] PKCS 1: RSA Cryptography Standard, 2004.

[3] Martin Abadi, Andrew Birrell, Michael Burrows, Frank Dabek, and Ted Wobber. Bankable postage for network services. In *ASIAN*, December 2003.

[4] E. Bangerter, J. Camenisch, and A. Lysyanskaya. A cryptographic framework for the controlled release of certified data. In *Security Protocols Workshop*, 2004.

[5] M. Bellare and O. Goldreich. On Defining Proofs of Knowledge. *CRYPTO*, 1992.

[6] J. Benaloh. Dense Probabilistic Encryption. *In Proceedings of the Workshop on Selected Areas of Cryptography*, 1994.

[7] A.J. Blumberg and R. Chase. Congestion pricing that respects “driver privacy”. In *ITSC*, 2005.

[8] A.J. Blumberg, L.S. Keeler, and A. Shelat. Automated traffic enforcement which respects driver privacy. In *ITSC*, 2004.

[9] D. Boneh, E-J. Goh, and K. Nissim. Evaluating 2-dnf formulas on ciphertexts. In *TCC*, 2005.

[10] F. Boudot. Efficient Proofs that a committed number lies in an interval. *EUROCRYPT*, 2000.

[11] G. Brassard, D. Chaum, and C. Crepeau. Minimum disclosure proofs of knowledge. In *JCSS, 37, pp. 156-189*, 1988.

[12] J. Camenisch, S. Hohenberger, and A. Lysyanskaya. Balancing accountability and privacy using e-cash. In *SCN*, 2006.

[13] J. Camenisch, S. Hohenberger, and A. Lysyanskaya. Balancing Accountability and Privacy Using E-Cash (Extended Abstract). *Security and Cryptography for Networks*, 2006.

[14] J. Camenisch and A. Lysyanskaya. A Signature Scheme with Efficient Protocols. *Security and Cryptography for Networks*, 2002.

[15] D. Chaum. Security without identification: transaction systems to make big brother obsolete. In *CACM 28(10)*, 1985.

[16] I. Damgard and E. Fijisaki. A Statistically-Hiding Integer Commitment Scheme Based on Groups with Hidden Order. *ASIACRYPT*, 2002.

[17] R. Dingledine, N. Mathewson, and P. Syverson. Tor: The second-generation onion router. In *USENIX Sec. Symp., USENIX Association*, 2004.

[18] C. Dwork. Differential privacy: A survey of results. In *TAMC 1-19*, 2008.

[19] E. Eide, P. H. Rubin, and J.M. Shepherd. *Economics of crime.* Now Publishers, 2006.

[20] M. H. Endrickson. The state of location-based social networking on the iphone, 2008.

[21] J. Eriksson, H. Balakrishnan, and S. Madden. Cabernet: Vehicular content delivery using wifi. In *MOBICOM*, 2008.

[22] A. Fiat and A. Shamir. How to Prove Yourself: Practical Solutions to Identification and Signature Problems. *CRYPTO*, 1986.

[23] foursquare. http://foursquare.com/, 2010.

[24] B. Gedik and L. Liu. Location privacy in mobile systems: A personalized anonymization model. In *25th IEEE ICDCS*, 2005.

[25] O. Goldreich. *Foundations of Cryptography*, volume II Basic Applications. Cambridge University Press, 2004.

[26] O. Goldreich, S. Micali, and A. Wigderson. How to Play Any Mental Game or A Completeness Theorem for Protocols with Honest Majority. *SIGACT*, 1987.

[27] O. Goldreich, S. Micali, and A. Wigderson. Proofs that yield nothing but their validity or all languages in np have zero-knowledge proof systems. In *Journal of the ACM, Volume 38, Issue 3, p.690-728*, July 1991.

[28] D. Goldschlag, M. Reed, and P. Syverson. Onion routing for anonymous and private internet connections. In *CACM, 42(2)*, 1999.

[29] S. Goldwasser, S. Micali, and C. Rackoff. The knowledge complexity of interactive proof-systems. In *Proceedings of 17th Symposium on the Theory of Computation, Providence, Rhode Island.*, 1985.

[30] S. Goldwasser, S. Micali, and C. Rackoff. The Knowledge Complexity of Interactive Proof-Systems. *Symposium on the Theory of Computation*, 1985.

[31] D. Goodin. Microscope-wielding boffins crack tube smartcard.

[32] E-ZPass Interagency Group. E-zpass.

[33] M. Gruteser and D. Grunwald. Anonymous usage of location-based services through spatial and temporal cloaking. In *ACM MobiSys*, 2003.

[34] M. Gruteser and B. Hoh. On the anonymity of periodic location samples. In *Pervasive*, 2005.

[35] 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. In *Mobisys*, 2008.

[36] B. Hoh, M. Gruteser, H. Xiong, and A. Alrabady. Enhancing security and privacy in trafc-monitoring systems. In *IEEE Pervasive Computing, 5(4):38-46*, 2006.

[37] B. Hoh, M. Gruteser, H. Xiong, and A. Alrabady. Preserving privacy in gps traces via uncertainty-aware path cloaking. In *ACM CCS*, 2007.

[38] 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. In *ACM SenSys*, 2006.

[39] Auto Insurance. Mile meter.

[40] J. Krumm. Inference attacks on location tracks. In *Pervasive*, 2007.

[41] S. Jha L. Kruger, E. Goh and D. Boneh. Secure function evaluation with ordered binary decision diagrams. In *ACM CCS*, 2006.

[42] H. Li, R. Guensler, and J. Ogle. An analysis of morning commute route choice patterns using gps based vehicle activity data. *Transportation Research Record*, 1926:162–170, 2005.

[43] T. Litman. London congestion pricing. *Victoria Transport Policy Institute*, 2006.

[44] A. Lysyanskaya, R.L. Rivest, A. Sahai, , and S. Wolf. *Pseudonym systems.* Springer, 2000.

[45] D. Malkhi, N. Nisan, B. Pinkas, and Y. Sella. Fairplay - a secure two-party computation system. In *USENIX Sec. Symp., USENIX Association*, 2004.

[46] U. Maurer. Unifying Zero-Knowledge Proofs of Knowledge. *AFRICACRYPT*, 2009.

[47] Mobile millennium. http://traffic.berkeley.edu/.

[48] M. Mun, S. Reddy, K. Shilton, N. Yau, P. Boda, J. Burke, D. Estrin, M. Hansen, E. Howard, and R. West. Peir, the personal environmental impact report, as a platform for participatory sensing systems research. In *Seventh Annual Intl. Conf. on Mobile Systems, Applications and Services (Mobisys 2009)*, 2009.

[49] M. Naor and O. Reingold. Number-theoretic constructions of efficient pseudo-random functions. In *Journal of the ACM, Volume 51, Issue 2, p. 231-262*, March 2004.

[50] California Department of Transportation. Caltrans guide for the preparation of traffic impact studies.

[51] U.S. Department of Transportation. Intellidrive.

[52] Bureau of Transportation Statistics. National household travel survey daily travel quick facts.

[53] T. P. Pedersen. Non-interactive and information-theoretic secure verifiable secret sharing. In *Springer-Verlag*, 1998.

[54] Sarah Perez. Location-Based Social Networks: Delightful, Dangerous or Somewhere in Between?, 2010.

[55] R. A. Popa, H. Balakrishnan, and A. J. Blumberg. VPriv: Protecting Privacy in Location-Based Vehicular Services. *USENIX Security*, 2009.

[56] G. Poupard and J. Stern. Security Analysis of a Practical "On The Fly" Authentication and Signature Generation . *EUROCRYPT*, 1998.

[57] S. Rass, S. Fuchs, M. Schaffer, and K. Kyamakya. How to protect privacy in floating car data systems. In *Proceedings of the fifth ACM international workshop on VehiculAr Inter-NETworking*, 2008.

[58] P.F. Riley. The tolls of privacy: An underestimated roadblock for electronic toll collection usage. In *Third International Conference on Legal, Security + Privacy Issues in IT*, 2008.

[59] Claudio Schapsis. Location based social networks links. http://bdnooz.com/lbsn-location-based-social-networking-links/.

[60] C. P. Schnorr. Efficient Identification and Signatures for Smart Cards. *CRYPTO*, 1989.

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

[62] Michael Walfish, J.D. Zamfirescu, Hari Balakrishnan, David Karger, and Scott Shenker. Distributed Quota Enforcement for Spam Control. In *3rd USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, San Jose, CA, May 2006.

[63] A. C. Yao. Protocols for secure computations (extended abstract). In *FOCS*, 1982: 160-164.

[64] G. Zhong, I. Goldberg, and U. Hengartner. Louis, Lester and Pierre: Three Protocols for Location Privacy. *7th Privacy Enhancing Technologies Symposium*, 2007.