VPriv: Protecting Privacy in Location-Based Vehicular Services

Raluca Ada Popa and Hari Balakrishnan
Computer Science and Artificial Intelligence Laboratory, M.I.T.

Andrew Blumberg
Department of Mathematics
Stanford University

(Part of the CarTel project http://cartel.csail.mit.edu/)

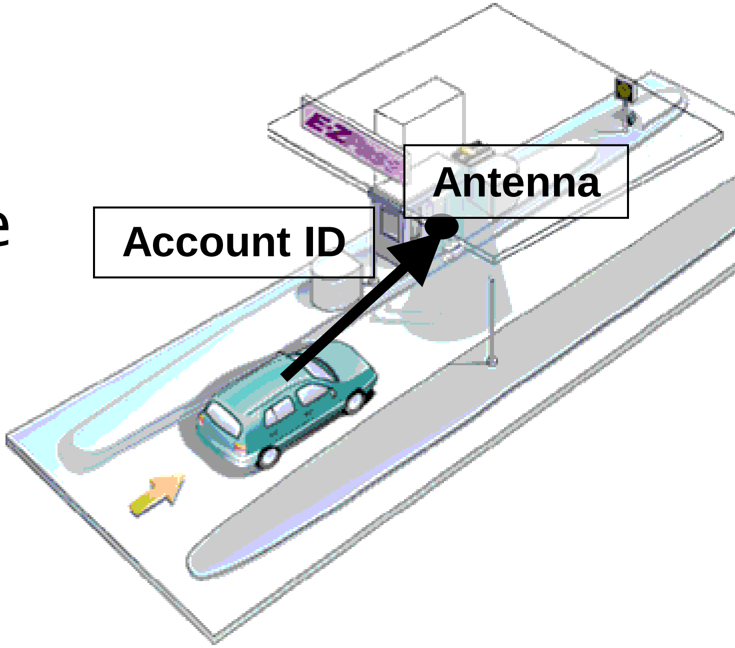
Motivation

- Location-based vehicular services are being increasingly adopted:
  - Automated toll collection (E-ZPass), traffic law enforcement, statistics collection
  - Insurance pricing based on driver behavior
- Promises efficiency, better driver experience, safety, revenue

Serious threat to the **locational privacy** of drivers!

Example: E-ZPass

- Antenna reads account ID, knows time, location
- A centralized server can assemble a driver's path
- Civil cases used driver path from E-ZPass data



➡ VPriv: a system for preserving privacy

VPriv

- Observation: Most vehicular services are *functions* over *time-location tuples*
- Compute functions on drivers' time-location tuples without revealing any information other than result

- Perform computations in zero-knowledge
  - Secure multi-party computation
- VPriv designed from scratch
- Efficiency through homomorphic encryption
- Applicable to sum of cost functions

Outline

- Motivation
- Model
- Architecture
- Protocols
- Enforcement
- Evaluation
- Conclusion

Model

- Two parties: car/driver and server
  - Driver is not trusted (transponder entirely not trusted)
  - Server is trusted to run protocol, but attempts to violate privacy
- *F* is a function to compute on driver's path
- Cars' transponders periodically generate tuples:

  <tag, time, location>
  - Tag is *random* and changing for privacy
  - Sent to server while driving or at end of month

Goals

- Correctness
- Locational Privacy
- Efficiency: important for deployment

Locational Privacy

VPriv

1. Database of <tag, time, location>
2. Client-server interaction during computation of *F*
3. Result of *F*

⇔

Oracle

1. Database of <time, location>
2. Result of *F*

- To prevent information being inferred from oracle database
  - Upload tuples only when enough mixing (Hoh et *al.*, 2008)

VPriv's Architecture

Two components:

1. Secure multi-party computation
   - Compute *F* on car's path
2. Enforcement scheme
   - Ensure clients abide by protocol

Applications

- Usage-based tolls
  - What is the toll a driver has to pay based on his path?
- Speeding tickets
  - Did the driver ever travel faster than 65MPH?
- "Pay-as-you-go" insurance premiums
  - How many minutes did the driver travel over the speed limit?
  - Did the driver travel through dangerous areas?

Crypto Tools

- Random function family: for k random, f_k looks random
- Commitment scheme
  - To commit to x, Alice computes $(c[x], d[x])$
  - Sends $c[x]$ to Bob; Bob cannot guess x
  - Later, Alice opens $c[x]$ by providing x and $d[x]$; cannot provide other x
  - Homomorphism:

$$c[x_1] \cdot c[x_2] = c[x_1 + x_2],\ d[x_1 + x_2] = d[x_1] + d[x_2]$$

Notation

- $\{v_i\}$: set of random tags of a ‘**v**’ehicle
- $\{s_j\}$: set of all tags seen at the ‘**s**’erver
- t_j : ‘**t**’oll associated with the tuple with tag s_j
  - $< s_j =$ 142, 4:21PM, GPS for Sumner Tunnel>, $t_j =$ \$3.5
- COST: total toll

Protocol

- Registration
  - Client chooses random tags, v_i, and a random function, k
  - Commits to $f_k(v_i)$ and k (sends $c[f_k(v_i)], c[k]$ to server)
- Driving
  - Uploads <v_i, *time, location*>
- Reconciliation
  - Using t_j from server, client computes the result of *F*
  - Server challenges the client to verify result
  - Detection probability ≥ ½ per challenge
  - Detection probability exponential in # challenges
    - (e.g. 10 challenges, 99.9% probability)

Reconciliation (cont'd)

- Tolling protocol
  - Server computes toll, t_j, for every tuple
  - Sends driver all pairs $\langle s_j, t_j \rangle$ for $t_j > 0$
  - Client computes total toll, COST

Challenge Phase

Server

$c[f_k(s_j)]$, $c[t_j]$ ←

Challenge 0: open $c[f_k(s_j)]$ and $c[f_k(v_i)]$ →

Challenge 1: open $c[f_k(s_j)]$ and $c[t_j]$; show f_k

Client

- Challenge 0: *assuming commitments are correct, verify COST*
  - Compute $\prod_{j:\exists i, f_k(s_j)=f_k(v_i)} c[t_j]$
  - Check it is a commitment to COST
- Challenge 1: *assuming COST is correct, verify commitments*
  - Chec $c[f_k(s_j)]$, $c[t_j]$ are correct

Why does it work?

- Correctness
- Soundness
  - Malicious client: commitments or *COST* are incorrect
- Locational privacy:
  - Challenge 0: reveal $f_k(v_i)$, but do not reveal f_k
  - Challenge 1: provide f_k, but do not decommit $c[f_k(v_i)]$

Related Protocols: Speeding

- Two consecutive tuples use same tag
  - Server computes speed between them
- Adjust tolling protocol
  - Server assigns cost of 1 to tuples over speed limit
- Speeding tickets: COST $\geq$ 1
- Insurance premiums
  - Number of speedups: COST

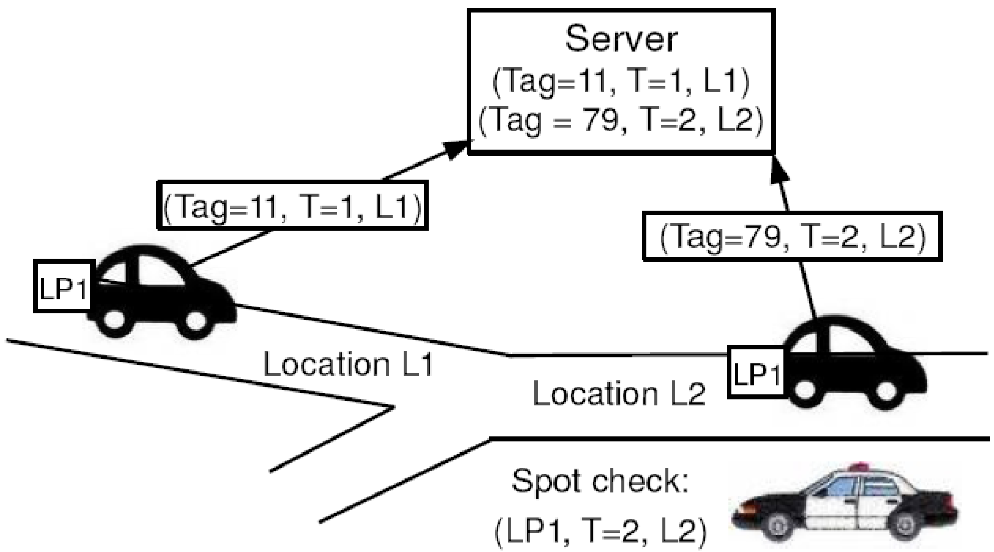
Enforcement

- Misbehaving clients:
  - Turn off transponder device
  - Use different tags
  - Modify location

Random spot checks

Random spot checks

- Police cars/cameras
- Record *<license plate, time, location>*
- Check for consistency with server's database



➡ General, applicable to all functions

Example Attack

- Client reneges some of his tags

1. Clients inform server which commitments from registration correspond to tags used while driving
2. Client downloads set of tuples from server and claims that all tags from driving are included
3. All spot checks collected are now checked for consistency; driver shows tuples corresponding to spot checks from driving; these tuples should have tags that are among the ones in Steps 1 and 2

→ If client reneged a tag in Steps 1 or 2, spot check fails

Outline

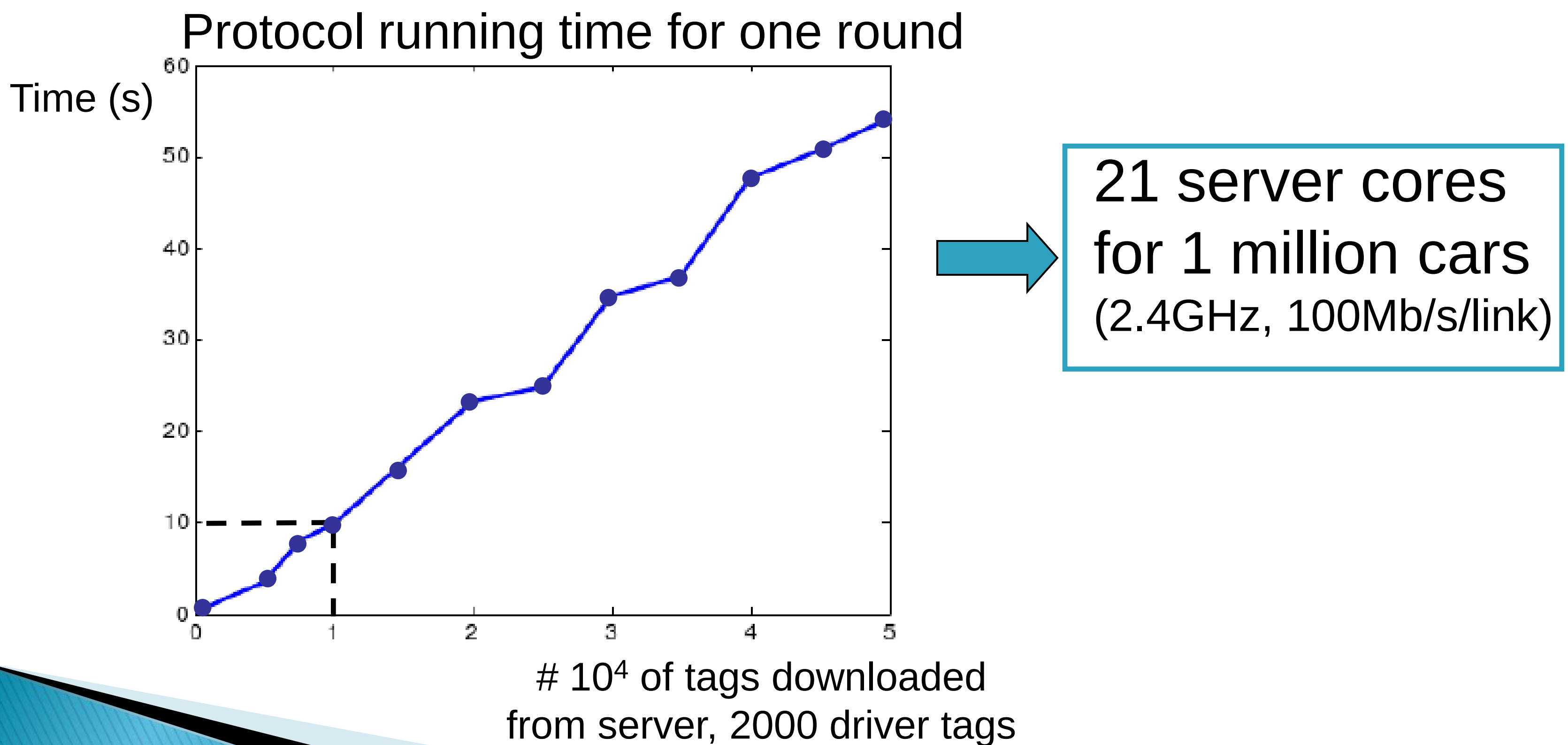
- Motivation
- Model
- Architecture
- Protocols
- Enforcement
- Evaluation
- Conclusion

Evaluation

- Tolling protocol, C++
- Linear in # of driver tags and tags downloaded from server
- Tradeoff privacy vs. efficiency

Implementation

- Registration and reconciliation
- 10 rounds, 10,000 tuples: ~100s running time/month



Comparison to Fairplay

- General purpose compiler for secure multi-party computation
- Implemented a simplified toll calculation
- Ran out of 1GB of heap space for 75 tuples, compiling and running > 5 min

➡ About *three* orders of magnitude slower than VPriv

Enforcement

- *Effectiveness similar to driving without a license plate*
- Detection probability is exponential in # of spot checks
  - E.g. 1 spot check/500 mins, driver detected with 95% in less than 10h
- Penalty reduces incentives
  - 1 spot check in 1000 mins, after 1.5h, detected ~10%
- Each driver spot checked about *1-2 times* a month

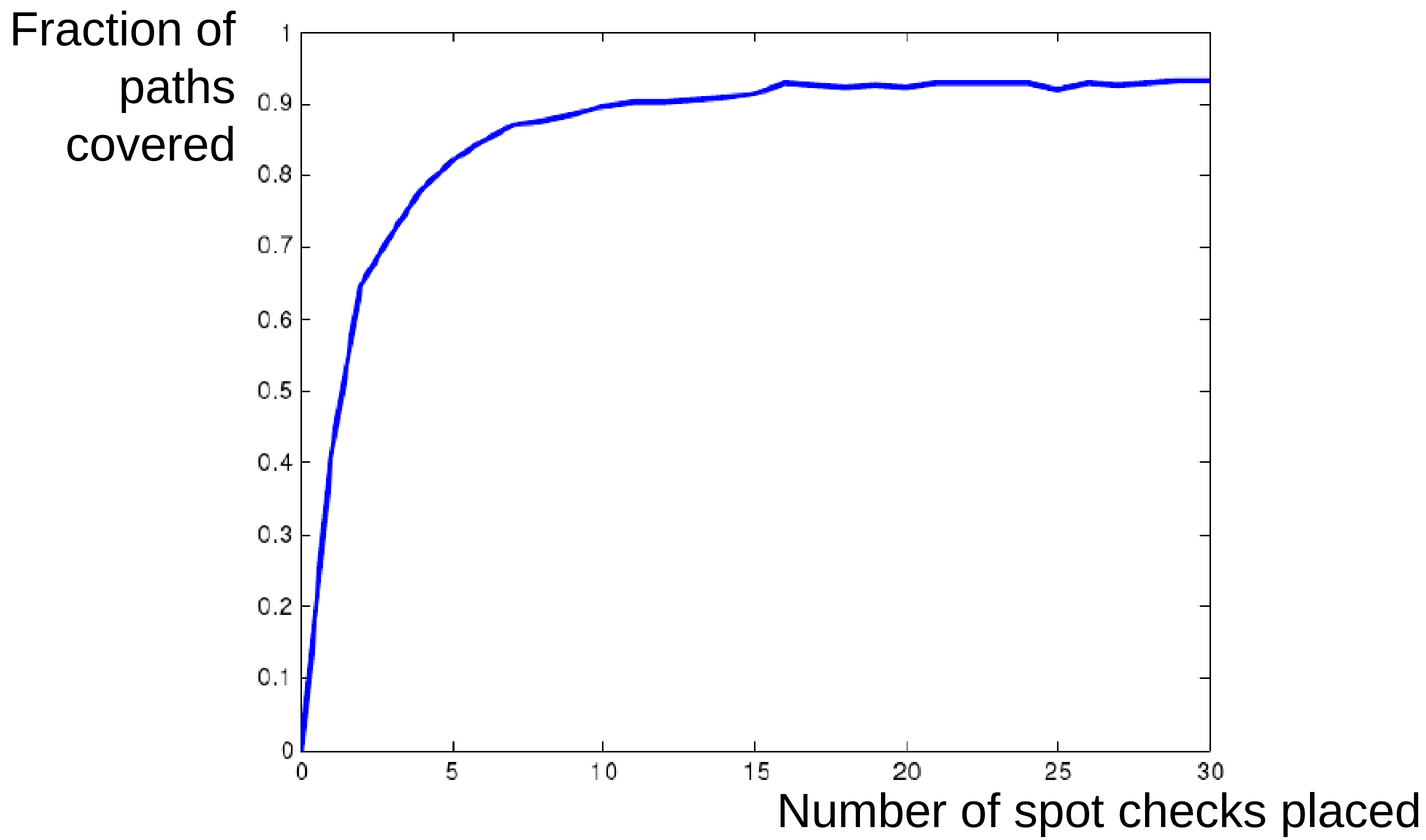
➡ Practical

➡ Privacy not affected

Simulation

- CarTel traces (*Hull*, 2008): 27 taxis in Boston area during year 2008, 4826 one-day paths
- *Training phase:* Extract 1% (~300) popular places during each month
- *Testing phase:* Place spot checks randomly at these places and record # of one-day paths observed

Simulation Results



- 15-20 spot checks, 90% paths covered (out of 4826)

Related Work

- *Blumberg et al.*, 2005
  - Use multi-party secure computation as a black box, no resilience to physical attacks
- E-cash (*Chaum,* 1985)
  - Not general approach, no enforcement
- Privacy in social networks (*Zhong*, 2007)
  - Specific point in polygon problem
- K-anonymity (*Sweeney*, 2002)
- Differential privacy (*Dwork*, 2006)
- Floating car data (*Rass*, 2008)

Conclusions

- Efficient protocol for preserving driver privacy
  - Wide class of vehicular services: tolling, speeding
- General and practical enforcement scheme
  - Spot checks

- Thank you!