# SPEcTRe: Spot-checked Private Ecash Tolling at Roadside

Jeremy Day
School of Computer Science
University of Adelaide*
Adelaide, Australia
jerday@gmail.com

Yizhou Huang
Cheriton School of
Computer Science
University of Waterloo
y226huan@uwaterloo.ca

Edward Knapp
Dept. of Combinatorics &
Optimization
University of Waterloo
eknapp@uwaterloo.ca

Ian Goldberg
Cheriton School of
Computer Science
University of Waterloo
iang@cs.uwaterloo.ca

## Abstract

Traditional stop-and-pay toll booths inconvenience drivers and are infeasible for complicated urban areas. As a way to minimize traffic congestion and avoid the inconveniences caused by toll booths, electronic tolling has been suggested. For example, as drivers pass certain locations, a picture of their licence plate may be taken and a bill sent to their home. However, this simplistic method allows the administrator of the system to build a dossier on drivers. While this may be an attractive feature for law enforcement, a society may not wish to trust the tolling agency with such detailed information. We present SPEcTRe, a suite of protocols to maintain driver privacy while ensuring that tolls are accurately collected. Existing protocols for privacy-preserving electronic toll pricing suffer from computational challenges and require an undesirable amount of location data to be collected. We present two schemes: the spot-record scheme, which requires the same amount of location data exposure as prior privacy-preserving schemes, but runs much faster, and the no-record scheme, which collects *no location information* from honest users and is still able to run efficiently.

## 1. INTRODUCTION

Electronic Toll Pricing (ETP) aims to improve road tolling by collecting toll fares electronically and without the need to slow down vehicles. In most ETP schemes, drivers are charged periodically based on the locations, times, distances or durations travelled. ETP is sometimes used to reduce traffic congestion as vehicles tend to avoid peak drive times due to the extra fees associated with them, such as with the London congestion charge [13]. Usually, these schemes require the use of an *on-board unit* in the vehicle. As opposed to traditional stop-and-pay tollbooths, ETP schemes can be used in a more pervasive manner, potentially making, for example, all roads in a downtown core into toll roads.

Many ETP schemes are currently deployed, such as e-TAG in Australia [11] and E-ZPass in the United States [19]. Singapore was the first to implement an ETP system to reduce congestion in 1998 [6]. Although these systems are efficient, they require a great deal of knowledge regarding driving habits in order to operate correctly. In particular, these systems store time, location, and identity, making it easy to build a profile of a driver's daily travel. In the United States, E-ZPass records can be obtained by court order in several jurisdictions for use in civil matters such as divorce [16]. Given that a high level of privacy can be obtained using traditional stop-and-pay toll booths, an ETP protocol that maintains a comparable level of privacy would be ideal.

We propose two ETP schemes in which tokens are broadcast periodically from the driver's vehicle. If a driver is caught not broadcasting a valid token, the driver is appropriately penalized. In this manner, we ensure that drivers behave honestly at all times, so long as they cannot discern when someone is monitoring their broadcasts. In our first scheme, we offer privacy guarantees similar to those offered by other privacy-preserving schemes. In particular, the central authority only learns the location data corresponding to times and places where the vehicle is physically observed. In our second scheme, we examine the cost of a system that does not require the collection of *any* private data of honest drivers. Both of these schemes rely on simple primitives such as RSA Full Domain Hash [9] and Chaum's ecash [7]. This is in contrast to the comparatively expensive zero-knowledge proofs and secure two-party computations needed by other ETP protocols, such as VPriv [18], PrETP [2], and Milo [15].

### 1.1 Our Contributions

We present SPEcTRe, a suite of protocols for electronic toll pricing. The SPEcTRe spot-record scheme records time-location information where drivers are physically observed. This is comparable to other schemes, but runs much faster, requiring only a modest amount of computational power to support one million vehicles. All the state-of-art ETP protocols including our spot-record scheme keep records of a small amount of location-time tuples in order to combat dishonest drivers. However, the SPEcTRe no-record scheme stores *no private information* of drivers, while still being capable of detecting cheaters.

### 1.2 Organization

The remainder of this paper is organized as follows. In section 2, we discuss other schemes that provide privacy-preserving electronic toll pricing. In section 3, we describe various cryptographic primitives that we use in SPEcTRe. Our spot-record-based scheme is outlined in section 4 and

*Work partially done while at the University of Waterloo.

our no-record scheme is given in section 5. We compare SPEcTRe with previous results in section 6. In section 7, we describe some of the hardware required to realize SPEcTRe. We give some results on the performance of SPEcTRe in section 8. Finally, we give concluding remarks in section 9.

## 2. RELATED WORK

Recently, much work has been done to build privacy-preserving schemes for electronic toll pricing and other related driving problems. For instance, work has been done to extend the problem to insurance pricing and speeding detection. Popa et al.'s VPriv [18] provides a practical protocol for computing path functions for several driving-related problems while maintaining a high level of privacy. The protocol uses a secure two-party computation to make sure drivers cannot cheat on the total tolling price if they do not cheat on the tokens they have uploaded. To ensure users upload the correct data, the authority is required to randomly record some {license plate, location, time} tuples, and then challenge drivers with these records during payment. It is important that the number of random observations, or "spot-checks", are kept to a moderate amount to maintain drivers' privacy. A measurement is performed to show that a modest hardware infrastructure can easily serve one million vehicles. VPriv is one of several recent papers that attempt to solve the problem of location privacy through the use of zero-knowledge proofs and secure multi-party computation.

Published a year after VPriv, Balasch et al. propose PrETP [2], a scheme with similar goals. They define a cryptographic protocol, Optimistic Payment, and prove it secure using zero-knowledge proofs and the RSA assumption. Unlike VPriv, they do not use secure two-party computations. Instead, the Toll Charger requires homomorphic commitments from clients, and the Toll Service Provider asks clients to open commitments to certain location-time tuples corresponding to its random spot-checks. For each client, only the total payment amount and location-time tuples in the physical vicinity of random spot-checks are revealed. They construct an on-board unit and analyze their system from a security, legal, performance, and cost perspective. Based on PrETP, Meiklejohn et al.'s Milo [15] furthermore considers the possibility that drivers may collude to learn the spot-checking locations. By utilizing blind identity-based encryption, these locations are not revealed to drivers.

The use of ecash to pay for driving has been suggested by Chaum and others [3, 8, 14] but the concept has not been used in a pervasive electronic tolling scheme until now. In particular, it is merely suggested that ecash replace physical currency. However, we have found that we can improve on this naive implementation by reducing the number of tolling points while maintaining driver honesty and privacy.

In VPriv, PrETP and Milo, spot-checks are necessary to uncover cheating drivers. SPEcTRe maintains the same level of privacy as these other schemes for drivers, in the sense that in all these schemes, location-time information of a vehicle is only revealed at spot-checking points. We also offer the same amount of security. Cheating drivers are detected, and a dishonest central authority cannot learn any more information than they do by spot-checking.

## 3. CRYPTOGRAPHIC PRIMITIVES

In this section, we describe various protocols which underlie SPEcTRe.

### 3.1 Signatures

The RSA Full Domain Hash signature scheme [9] is secure relative to the *RSA assumption*, the difficulty of finding $e$-th roots modulo a composite integer. The RSA assumption is conjectured to be equivalent to factoring. The scheme consists of three algorithms, `Generate`, `Sign`, and `Verify`.

`Generate` constructs an RSA modulus $N$ and integers $e$, $d$ such that $ed \equiv 1 \pmod{\varphi(N)}$. The algorithm returns a public key $(e, N)$ and a private key $(d, N)$. The algorithm publishes a hash function $H\colon \{0,1\}^* \to \mathbb{Z}_N$.

`Sign` takes as input a message $m$ and returns the signature $\sigma = (H(m))^d$.

`Verify` takes as input a message $m$ and a signature $\sigma$. The signature is valid iff $\sigma^e = H(m)$.

The `Generate` and `Sign` algorithms are run by the signer. The `Verify` algorithm can be run by anyone in possession of the corresponding public key. We remark that although the private key $(d, N)$ is sufficient to use the scheme, in practice, keeping the factors of $N$ as part of the private key speeds up the `Sign` algorithm.

### 3.2 ecash

Chaum [7] introduced the concept of ecash. The essential idea is that a user can purchase a single interaction with a server the end result of which is a cryptographic *coin* in the possession of the user which can only be created by the server. Presuming the server demands a fixed amount of a given currency to create a coin and presuming a coin can be redeemed for the amount in question, the coin may be treated as though it were the underlying currency.

The attractiveness of Chaum's scheme lies in the privacy guarantees. In particular, given two coins created in two separate interactions with the server, it is information-theoretically impossible to relate the coins to the respective events which created each coin.

We present an instantiation which produces RSA Full Domain Hash [9] signatures but the result can be generalized to other signature schemes. Chaum's scheme consists of the algorithms `Generate`, `Commit`, `Sign`, and `Open`.

`Generate` returns a public key $(e, N)$, a private key $(d, N)$, and a hash function $H\colon \{0,1\}^* \to \mathbb{Z}_N$ as detailed in section 3.1.

`Commit` takes as input a message $m$. The algorithm selects $x \in \mathbb{Z}_N$ uniformly at random and returns $c = x^e \cdot H(m)$.

`Sign` takes as input a commitment $c$ and returns $\gamma = c^d$.

`Open` takes as input a signed commitment $\gamma$ and the corresponding random value $x$. The algorithm returns the signature $\sigma = \gamma \cdot x^{-1}$.

The algorithms `Generate` and `Sign` are run by the signer, who holds the private key, while the remaining algorithms are run by the client, who holds the public key.

Finally, given the public key $(e, N)$, anyone can run, on input $(m, \sigma)$, the `Verify` algorithm which returns the Boolean value $\sigma^e \stackrel{?}{=} H(m)$.

The security of Chaum's scheme can be proven equivalent to solving the One-More-RSA-Inversion problem [4]. In practice, it is generally assumed that this is equivalent to the RSA problem but currently it is only known that the One-More-RSA-Inversion problem can be reduced to the RSA problem.

### 3.3 Pairings and BLS

Let $G_1$, $G_2$, $G_T$ be groups of order $r$. An asymmetric bilinear pairing $e\colon G_1 \times G_2 \to G_T$ is a map which is linear in each coordinate (under the group operations). We describe how to use a pairing to implement a (blind) signature scheme. The advantage of the pairing-based schemes is that compared to RSA Full Domain Hash, signatures are smaller and signing is more efficient at higher security levels.

The BLS signature scheme [5] is defined as follows. As global public parameters, generators $g_1 \in G_1$, $g_2 \in G_2$, and a hash function $H\colon \{0,1\}^* \to G_1$ are selected. A signer selects a random integer $a$ in the interval $[1, r-1]$ as his private key and publishes the corresponding public key $A_1 = g_1^a$, $A_2 = g_2^a$. To sign the message $m$, the signer computes the signature $\sigma = H(m)^a$. To verify the signature $\sigma$ of the message $m$ signed under the public key $(A_1, A_2)$, the driver checks that $e(H(m), A_2) \stackrel{?}{=} e(\sigma, g_2)$. To save space, the $y$-coordinate of the signature $\sigma$ can be thrown away; unlike the generic situation for elliptic curve point compression, for BLS, not even one bit need be kept [17]. This comes at a slight extra cost for verification, as that coordinate will need to be recomputed, but this extra cost is negligible compared to the pairing operations (over two orders of magnitude cheaper).

In order to obtain blind signatures, a driver wishing to have the message $m$ signed can give $g_1^x H(m)$ to the signer for some random $x$, receive back $\gamma = (g_1)^{xa} H(m)^a$ from which the signature $\sigma = \gamma \cdot A_1^{-x}$ can be computed. This allows us to implement SPEcTRe using pairings instead of RSA.

## 4. SPOT-RECORD SCHEME

There are four main components of a tolling scheme. A *registration server* registers drivers, possibly providing them with an on-board unit. A *driver* drives on toll roads and verifiably pays for driving either during or after transit. A *verifier* occasionally monitors the drivers as they drive. A *payment server* ensures that drivers pay for the amount of driving they perform. We could expect that the registration server, payment server, and verifier are managed by the same entity. The goal is to protect the privacy of the driver while enforcing that the driver correctly pays for his driving. Specifically, we concern ourselves with *location privacy*, which is the linkability of the times and places a particular driver has travelled.

In our scheme, the driver obtains *tokens* from the registration server at the beginning of some time period (for example, a month). While driving, the driver spends the tokens by broadcasting them. At the end of the time period, the driver interacts with the payment server to redeem unused tokens and pays for the tokens used.

We present SPEcTRe using the RSA Full Domain Hash signature scheme. However, the BLS signature scheme can be used as well. The tolling protocol consists of three phases: *registration*, *driving*, and *reconciliation*. During registration, the driver is assigned a random private identity $i$ and a set of tokens of the form $(r, \sigma)$, where $\sigma = H(r, i)^d$ and $r$ is a random string. The identity $i$ should correspond to the license plate of the driver's vehicle but the correspondence should be kept private from third parties.

While driving, the driver broadcasts tuples $(r, \sigma)$, switching tuples at predefined intervals. In order to ensure drivers are broadcasting, a verifier secretly monitors drivers at random locations and times. For each driver, the verifier takes a picture of the driver's licence plate and records all tuples $(r, \sigma)$ which are being broadcast. No further work needs to be done by the verifier other than recording and storing this data.

For each license-plate photograph, there is a set of records $\{r_j, \sigma_j\}_j$ that were recorded in the vicinity. The *registration server* derives an identity $i$ from the photograph and ensures that $\sigma_j^e = H(r_j, i)$ for some index $j$, forming a tuple $(i, r_j, \sigma_j)$ for each photograph. This server also checks for double spending by ensuring no identity uses the same random string $r_j$ in two different locations. If no signature is found for a given photograph or a signature appears twice, it is concluded that the driver is cheating.

Finally, during payment, the driver submits all tuples $(r, \sigma)$ which were not spent. The payment server verifies that none of these tokens were detected and collects payment for those tokens not submitted. Alternately, full payment for all tokens can be made at registration time and a refund for the submitted tokens can be received during reconciliation.

If the verifier is capable of taking photographs and detecting tokens in a stealthy manner, then the driver must always spend tokens while driving to avoid penalty. Since a private random identity is chosen for each licence plate, tuples cannot be linked to each other without knowledge of the identity. Since the identity is not broadcast as part of the token, a third party cannot determine the identity corresponding to a given tuple. Location data is only collected during spot-checks. This is similar to VPriv, PrETP, and Milo, and is a vast improvement over schemes implemented today, such as E-ZPass.

## 5. NO-RECORD SCHEME

In the preceding scheme, a small amount of location-time information about honest drivers is recorded in order to combat cheating drivers. However, in an ideal privacy setting, we want to collect no information about honest drivers at all, but still be able to detect dishonest drivers. Our no-record (NR) scheme is an exploration of this possibility.

To accomplish this, we require the additional engineering capability that the spot-checking verifiers be able to determine *which* car is broadcasting which token, through the use of directional antennas or triangulation, for example.

Initially, the registration server runs the ecash `Generate` algorithm and gives the public key to all other components in the scheme.

From the perspective of the driver, our scheme consists of three algorithms: an interactive `Create` algorithm, the `Verify` algorithm, and the `Payment` algorithm. Contained with the `Create` algorithm is the collection of ecash algorithms. The driver selects a random string $m \in_R \{0,1\}^*$ and a random integer $x \in_R \mathbb{Z}_N$. The driver sends $c = x^e H(m)$ to the registration server and obtains $c^d$ from the server. From $c^d$, the driver can compute the signature $\sigma = x^{-1}c^d = H(m)^d$. The `Create` algorithm is described in Figure 1. The driver runs the `Create` algorithm $n$ times to get $n$ tokens from the registration server.

| **User** | | **Registration Server** |
|---|---|---|
| $m \in_R \{0,1\}^*$ | | |
| $x \in_R \mathbb{Z}_N$ | | |
| $c = x^e H(m)$ | $\longrightarrow$ | c |
| $\gamma$ | $\longleftarrow$ | $\gamma = c^d$ |
| $\sigma = \gamma \cdot x^{-1}$ | | |

**Figure 1: The creation of one token**

While driving, the driver broadcasts tokens $(m, \sigma)$ at a predetermined rate. The `Verify` algorithm is run by the verifier as these tokens are received. If the `Verify` algorithm indicates that a signature is invalid, or if the driver is not broadcasting tokens at all, the driver is held accountable in some manner *at the time the algorithm is run.* In practice, the verifier will be associated with a camera or police presence which can take appropriate measures.

After a predetermined period of time (eg. monthly), the driver's tokens are invalidated. At this time, the driver interacts with the payment server to redeem his unused tokens and pay for the service Invalidation can be done by using a different public key $(e, N)$ for each time period.

The spot-record scheme has the advantage that the verifier does not need to do any work online other than photographing license plates and recording tuples. The no-record scheme has the advantage that the amount of private data recorded is reduced but requires a more involved verifier in order to detect malicious activity, as outlined next in section 5.1.

## 5.1 Combating Double Spending

Malicious drivers have several opportunities to double spend their tokens. First, a single driver can use the same token more than once while driving. Second, a single driver can spend a token while driving and then attempt to redeem the token during payment. Third, two drivers can use the token at the same time in different locations. An individual verifier can maintain a sorted list of detected tokens and check new tokens against this list to combat double spending locally. If the verifiers periodically synchronize and ultimately report their results to the payment server, the first and second attacks can be combated. However, without constant communication between verifiers, two drivers using the same token at the same time cannot be detected.

In light of the fact that we need the verifiers to be synchronized to prevent double spending, we introduce a centralized server called the *verify server* which interacts with the verifier to combat double spending. After the verifier ensures the signature is correct, the message $m$ is sent to the verify server to be checked against other messages. The server indicates to the verifier if the token was double spent or not.

## 5.2 Reducing the Cost of Double-Spending Detection

Since verification is a very time-sensitive procedure, we would like to reduce the server-side latency as much as possible. We can search and insert into a B+ tree in time $O(\log n)$. With even a modest number of vehicles, the number of tokens and verification percentage results in very rapid growth in the size of the B+ tree. If $n$ were small enough to fit in memory, it would greatly reduce the server-side latency and eliminate the need for complicated database-management techniques. To this end, we show that by binding $m$ to a time or location, the double-spending detection cost can be reduced significantly.

We modify SPEcTRe as follows. During the `Register` algorithm, for each possible time $t$, the driver constructs $m = (H(r,t), H(i,s))$, where $r$ and $s$ are fixed-length bitstrings chosen uniformly at random. The driver obtains a signature on $m$ in the same manner as in the preceeding description of the algorithm with the final result being a set of tuples of the form $(t, r, i, s, \sigma)$. We suggest that the granularity of $t$ be 5 minutes for instance. If this granularity is too small, then as we will see in section 8, the performance of this scheme might suffer as the registration phase would turn out to be a bottleneck.

While driving, the driver selects the token $(t, r, i, s, \sigma)$ corresponding to the current time. The driver computes $b = H(i,s)$ and broadcasts $(t, r, b, \sigma)$. The verifier receives this tuple and rejects the token if $t$ does not correspond to the current time window. It computes the value $m = (H(r,t), b)$ and verifies the signature as $H(m) \stackrel{?}{=} \sigma^e$. Finally, the value $m$ is sent to a central server to ensure that the token is not being double spent.

During the time corresponding to $t$, the verify server needs to compare tuples $(t, r, b, \sigma)$ only to other tuples of the form $(t, r', b', \sigma')$ in order to detect double spending. If $n$ cars are being detected during the time period corresponding to $t$, the total time and space complexities for detecting double spending are $O(n \log n)$ and $O(n)$ respectively. Of particular note is the amount of space required. One million vehicles and a 160-bit value for $m$ gives a total memory cost of 20 MB.

Finally, the payment server collects all sorted time-based lists and merges them into a master sorted list. The driver computes $a = H(t, r)$ and submits a sorted list of tokens

**Table 1: A comparison of various schemes in terms of the complexities of each phase, security and privacy properties, and also whether real-time cheating detection (RCD) and real-time verifier communications (RVC) are required when drivers are driving on the road. $n$ is the number of tokens per driver. $u$ is the number of drivers. $k$ is the average number of tokens spent per driver. $r$ is the average number of detections per driver. $\rho$ is the granularity of the bound time (a small fraction), so $\rho^{-1}$ is the number of time slices for one month. All algorithms are per driver. ZKP stands for zero-knowledge proofs and SMC denotes secure-multiparty computation. Hash denotes the assumption of a preimage-resistant and collision-resistant hash function. Milo also includes an `Audit` phase which has been grouped with `Verify` (and thus unchanged, since $r < k$). Also note that the constants hidden behind the $O$ notation for schemes with ZKP and SMC are quite large, while for SPEcTRe, they are very small. Privacy means the extent to which data of honest users are unmonitored.**

| | `Verify` / driving | `Payment` / reconciliation | Security assumptions | Privacy | RCD | RVC |
|---|---|---|---|---|---|---|
| VPriv | $O(k)$ | $O(k)$ | RSA, ZKP, SMC | Partial | No | No |
| PrETP | $O(k)$ | $O(k + u \cdot r)$ | RSA, ZKP | Partial | No | No |
| Milo | $O(k)$ | $O(k)$ | RSA, ZKP, SMC | Partial | No | No |
| spot-record | $O(r)$ | $O(n \log(n))$ | RSA, Hash | Partial | No | No |
| Basic NR | $O(r \log(u \cdot r))$ | $O((n-k) \log(u \cdot r))$ | RSA, Hash | Full | Yes | Yes |
| $t$-bound NR | $O(r \cdot \rho \log(u \cdot r \cdot \rho))$ | $O((n-k) \log(u \cdot r))$ | RSA, Hash | Full | Yes | Yes |

to be redeemed of the form $(a, s, \sigma)$. Since the identity $i$ is fixed for all tuples, there is no need to submit it multiple times. The payment server computes $m = H(a, H(i, s))$ for each tuple, ensures the list is sorted by the $m$'s, runs the `Verify` algorithm on each tuple and ensures that there are no repeated tuples. Finally, the payment server checks the submitted list against the master list to detect if the driver is attempting to redeem spent tokens. Since the tokens that drivers are attempting to redeem are bound to their identity, they cannot be spent by two drivers. Hence, the submitted list need not be added to the master list. The driver pays for those tokens not returned to the payment server.

We will describe the security and privacy implications of this modification in the next section. We also remark that the entire $m$ need not be transmitted during the verify phase; we could transmit only the first few bits of $H(m)$. For example, if we expect to store $2^s$ tokens, if we sent only the first $2s+60$ bits of $H(m)$, we would experience collisions with probability $2^{-30}$.

## 5.3 Complexity

In Table 1, we give the complexities of our no-record scheme with and without time-bound tokens. The registration algorithms have identical complexity but the complexities of both `Verify` and `Payment` are greatly reduced with the addition of the time variable. In both algorithms, the bottleneck of computation is on the server side.

## 5.4 Probability of Detecting Double Spending

One of the most attractive aspects of SPEcTRe is that verifiers need not check all tokens. Presuming the drivers cannot detect the verifiers, they have no choice but to assume someone is listening and behave honestly. Let $\nu$ denote the fraction of spent tokens which are being detected by verifiers. Then the probability of detecting double spending during verification is $\nu^2$. The probability of detecting a driver attempting to redeem a spent token is $\nu$.

## 5.5 Security of Tokens

We require that it is difficult for drivers to create more tokens than are given to them during the registration protocol. The security of SPEcTRe is equivalent to the security of ecash, relying on the difficultly of the One-More-RSA-Inversion problem.

## 5.6 Privacy

During registration, it's clear that tokens are private in the information-theoretic sense since they are based on a standard ecash scheme. We argue that no relation can be ascertained between tokens revealed during different algorithms. In particular, for $m = H(a, b)$ with $a = H(t, r)$ and $b = H(i, s)$, since the images $a$ and $b$ are from a salted hash, it is computationally infeasible to relate $a$ to $t$ or $b$ to $i$.

# 6. COMPARISON WITH OTHER WORK

Table 1 shows the complexity of our schemes with various others. In the PrETP scheme, the complexity in the verify is due to the spot-checks, while in VPriv and SPEcTRe, the complexity is due to token broadcasting and processing.

Compared to previous work, our schemes rely on much more common security assumptions, such as the RSA problem and random oracles. Although the complexities of payment and reconciliation may appear much more expensive than previous work, our schemes are based on simpler primitives, and so the constants in the big-O notation are much smaller in our scheme, as evidenced by our measurements in section 8.

# 7. HARDWARE INFRASTRUCTURE

There are three basic hardware components in the verifying and payment phases of SPEcTRe: a transponder on each vehicle, token readers at various points along the road, and a centralized server that maintains the database to check double spending.

Commercial manufacturers are working on transponder devices which allow for reliable vehicle-to-roadside or vehicle-to-vehicle communications. The communication protocol

is layered [12], giving application-level developers flexibility in designing and implementing their own protocols. These communications are based on dedicated short-range communications [10]. We did a proof-of-concept experiment, broadcasting tokens in a moving vehicle driving at 40 km/h from a Nexus One smartphone, and they were reliably received on the roadside by another Nexus One.

Token readers could either be police cars moving around discreetly or some reader device on the roadside. Whatever those readers are, it is important that drivers are unable to determine if someone is monitoring their broadcasts.

The registration phase requires some server to sign tokens for clients so that transponders can broadcast these tokens. These operations can be carried out between a driver's personal computer and a server. A simple TLS-protected connection is sufficient.

## 8. PERFORMANCE MEASUREMENT

In this section we evaluate the runtime and storage requirements for the primitives of SPEcTRe. In our measurement, we let each client serially go through the three phases: registration, verification, and payment. The measurement is based on the no-record scheme, but with an extreme case where all the tokens broadcast are collected and even tokens not revealed during driving are added to the central database for simplicity.

The proof-of-concept implementation was written in C++ with a multithreaded registration phase. The code is based on a server-client model where each client, representing a vehicle, first gets tokens for the whole month, reveals half of the tokens, and then uses another half in the payment phase. For every token a client reveals, the server stores that token in the database to check double spending later. Our measurement does not utilize the knowledge of $t$ to speed up the payment phase for simplicity, and we add every token in the payment phase to the database, which actually reduces the speed, because with identities revealed in the payment phase, we can simply check double redemption for each individual client. However, we still found that we beat VPriv in terms of computation time given the same number of tokens used.

We only compare in detail the performance of our spot-record scheme with VPriv, because these two schemes share a similar hardware infrastructure. PrETP requires heavy real-time computation on the on-board unit, which would require specialized crytographic coprocessor when the key size goes to 2048 bits. PrETP also requires the incorporation of GPS data on each vehicle. In terms of computation requirements on the server side, for a segment size of 1 km and a key size of 2048 bits, PrETP takes 88.050 seconds to verify the payment of each individual driver [2]. SPEcTRe takes only around 16 seconds to both register and verify one month's worth of tokens for a security level of 128 bits by using pairing-based scheme described earlier, corresponding to a RSA key size of 3072 bits.

We used the C/C++ interface of SQLite to manage our database and OpenSSL libraries to implement the cryptographic primitives. The measurement was run on a laptop

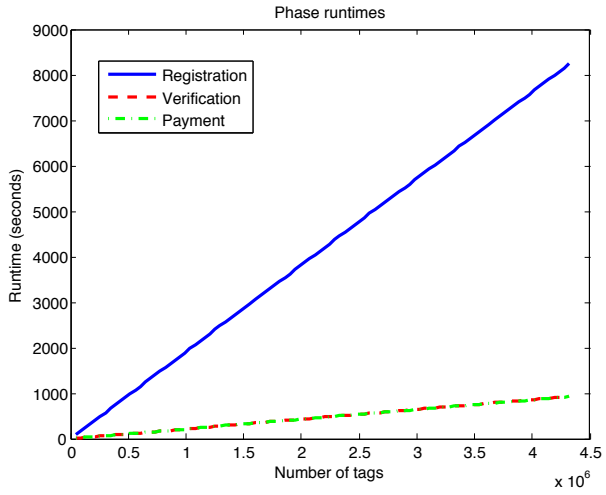


**Figure 2: Runtimes of each phase for many tokens**

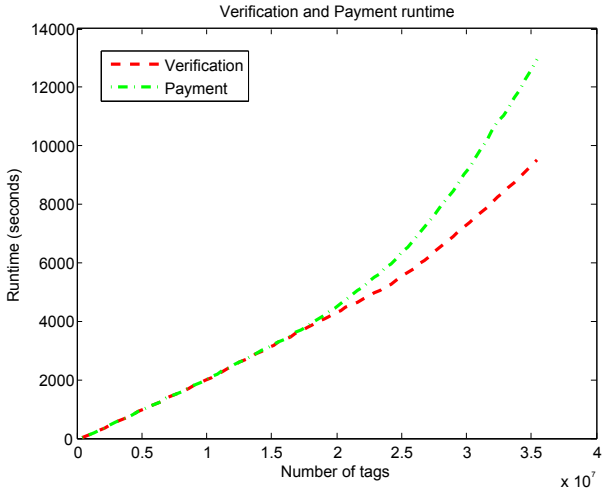


**Figure 3: Runtimes of verification and payment with 10 times as many tokens as Figure 2.**

with two 1.86 GHz cores and 2 GB RAM, with 32-bit 10.04 Ubuntu (lucid) installed. The client and the server were running on the same machine, so we were not simulating the network. However, we did estimate the communication cost and we will argue that our bandwidth requirement is not high.

Figure 2 shows our runtime for three phases, corresponding to different number of tokens used. Let us assume the average driver travels 18,000 km per year [18]. This equates to 40 hours of driving per month for each car. If drivers are required to reveal a new token every minute, and tokens are not bound to $t$, then the number of tokens for each client thus would be $40 \times 60 = 2400$ every month. The decision to reveal a new token once per minute was chosen to better compare with VPriv. If we are to bind $t$ to each token, the resolution of time should be chosen carefully so that the number of tokens issued would allow registration to finish in a reasonable amount of time.

There are some practical concerns here, however, some of which make SPEcTRe faster, while some of them make SPEcTRe slower.

One concern is that although the runtime for revealing and payment appears to be linear when the dataset is small, it should not always be the case when the database becomes larger. According to the observation of our experiment, SQLite organizes the database with hash tables. As the database becomes larger the linear performance would not be maintained due to collisions and reorganizations. In order to find out the performance for a larger dataset, we ran another measurement which generates random numbers and insert them into the database. The performance is shown in Figure 3, which shows that when the number of tokens grows to 34,992,000 (14,580 clients), the runtime for each insertion grows to twice as long as before.

There are also ways to make SPEcTRe faster. First, we can replace the RSA scheme with a pairing-based scheme. Since currently the bottleneck is the registration phase, specifically the blind signature computation, a pairing-based scheme would significantly improve our runtime. This is discussed in more detail in section 8.1. Second, the registration phase is also completely parallelizable and can be done in advance. For example, we may choose to allow registration for the next month. All that we need to ensure is that the server's key pair is regenerated every month. Third, we can also easily utilize the $i$ value so that we do not need to add tokens to the database during the payment phase since we only need to check that every client does not submit two identical tokens.

In order to hold the number of tokens generated in our experiment, namely 34,992,000 tokens, the size of the database is 544.9 MB. So in order to hold tokens for one million clients, we need $544.9 \times 1,000,000/14,580 = 37373.11$ MB, which is about 37 GB storage space. This is small for an average server. We can also save some space by not storing the entire token, but instead truncating the hash to a smaller size that makes collisions unlikely, as discussed earlier.

We also measure the communication cost. For the payment phase, the client will reveal every token that is not spent. Assume that a client does not spend any tokens at all, which is the worst case. With the RSA scheme, every token is $|H(t)| + |\sigma| = 256 + 3072 = 3328$ bits at a 128-bit security level. Thus, one client needs to transmit $2,400 \times 3328$ bits $= 998$ KB in a whole month. For the server, assume that there are 1 million clients who do not spend any tokens at all throughout the whole month. There will be 1 million $\times$ 998 KB which is approximates 998 GB of data transmitted. The bandwidth required for the server is only 0.385 MB/s. If we are to allow multiple prices by requiring more tokens to be revealed on expensive roads, these numbers would go up but they are still acceptable even if multiplied by 10. The pairing-based scheme with a security level of 128 bits has a signature length of 256 bits. In that case, the communication cost would be even less. The communication cost between verify servers is negligible.

We summed up the runtime for all three phases in Figure 4, which also compares SPEcTRe with VPriv. For 1800 clients

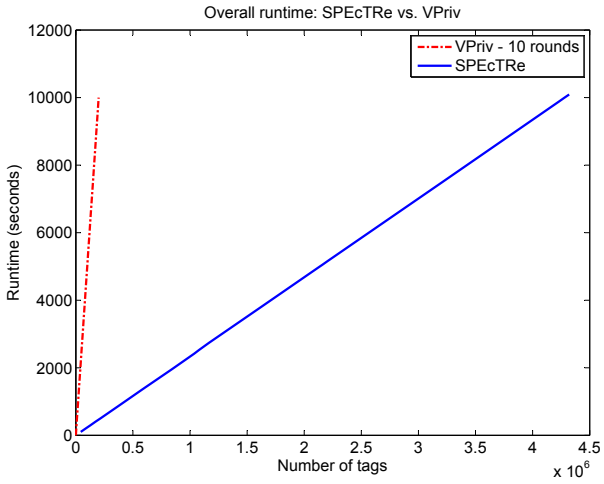


**Figure 4: Runtime comparison of our no-record scheme with VPriv.**

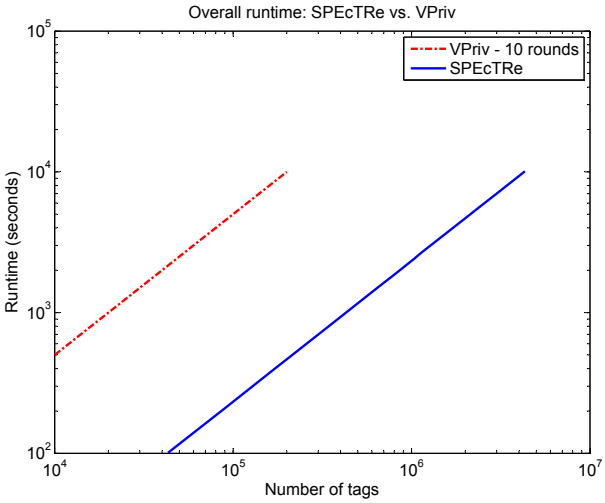


**Figure 5: Log-runtime comparison of our no-record scheme with VPriv.**

(4,320,000 tokens), SPEcTRe takes about 10,000 seconds. The runtime for VPriv grows linearly with the number of tokens, and VPriv requires 100 seconds for 2,000 tokens in a recommended setting of 10 rounds computation. In this sense, SPEcTRe is $4,320,000/(2,000 \times 100) = 21.6$ times faster. Figure 5 better demonstrates this speed difference. This does not take into account the fact that their measurement ran on a more powerful computer. With similar hardware, SPEcTRe runs much faster than VPriv.

## 8.1 BLS vs RSA

Registration is the slowest phase in the above measurement, as seen in Figure 2. By using a pairing-based signature scheme to sign and verify tokens, we are able to make the registration phase much faster. The performance of the BLS pairing-based scheme and the RSA scheme were measured on a 64-bit machine. We used relic-toolkit version 0.3.0 [1] to implement the pairing-based scheme, and OpenSSL to implement the RSA scheme, both at a 128-bit security level (256-bit curves for BLS and a 3072-bit modulus for RSA). With the computation power of a single core, the times are shown in Table 2.

**Table 2: The performance of the BLS and RSA signature schemes**

| | Sign | Verify | Signature size |
|---|---|---|---|
| BLS | 0.4 ms | 5.2 ms | 256 bits |
| RSA | 13 ms | 0.3 ms | 3072 bits |

As we can see, the pairing-based scheme is doing a much better job in signing blind signatures, while the RSA scheme is slow in signing but much faster in verification. In our no-record scheme, where drivers who are double-spending should be detected on the fly, faster verification reduces one possible bottleneck. However, one should also notice that even assuming 20 cars are passing by within one second, BLS verification only takes 104 ms, which may not be a great issue when compared to network delay. In our spot-record scheme, where double-spending detection can be done offline, verification time is a less critical factor. Given that a client doesn't reveal all his spent tokens to the verifier, it is optimal to utilize faster BLS signing to decrease the overall computation tasks on the server side. The pairing-based scheme requires a smaller signature, which lowers the communication cost at places where signatures are required. But this feature would not help much for the real-time communications between verifiers, because they do not need signatures to be exchanged in order to combat double-spending.

# 9. CONCLUSIONS

We presented two electronic toll pricing schemes that preserve privacy. When compared to other schemes, the spot-record scheme offers comparable privacy guarantees with standard security assumptions. No intensive nor interactive computation is required on the on-board unit. The no-record scheme has the property that no private location information of drivers needs to be recorded in order to detect dishonest drivers, however, more powerful verifiers would be required. Both of our schemes are compared with other systems, specifically VPriv. We offer the same amount of security, and even more privacy in the no-record scheme, while our proof-of-concept measurement indicates that they both run considerably faster than related works.

**Acknowledgements.** We would like to thank Kevin Bauer, Tariq Elahi, Angèle Hamel, Ryan Henry, Femi Olumofin, and Rob Smits for their helpful comments on drafts of this paper. A special thanks to Tariq Elahi, Rob Smits, and Dennis Zhuang for their assistance with the broadcasting experiment. We also gratefully acknowledge the funding support of NSERC and MITACS.

# 10. REFERENCES

[1] Aranha, D. F., and Gouvêa, C. P. L. RELIC is an Efficient LIbrary for Cryptography. `http://code.google.com/p/relic-toolkit/`.

[2] Balasch, J., Rial, A., Troncoso, C., Geuens, C., Preneel, B., and Verbauwhede, I. PrETP: Privacy-Preserving Electronic Toll Pricing. In *19th USENIX Security Symposium* (Washington, DC, 2010).

[3] Bangerter, E., Camenisch, J., and Lysyanskaya, A. A Cryptographic Framework for the Controlled Release of Certified Data. In *Security Protocols Workshop* (2004), pp. 20–42.

[4] Bellare, M., Namprempre, C., Pointcheval, D., and Semanko, M. The One-More-RSA-Inversion Problems and the Security of Chaum's Blind Signature Scheme. *Journal of Cryptology 16* (2003), 185–215.

[5] Boneh, D., Lynn, B., and Shacham, H. Short Signatures from the Weil Pairing. *Journal of Cryptology 17* (2004), 297–319.

[6] Cervero, R. *The Transit Metropolis: A Global Inquiry*. Transportation / Planning. Island Press, 1998, ch. 7.

[7] Chaum, D. Blind Signatures for Untraceable Payments. In *CRYPTO'82* (1982), pp. 199–203.

[8] Chaum, D. Security Without Identification: Transaction Systems to Make Big Brother Obsolete. *Commun. ACM 28* (October 1985), 1030–1044.

[9] Coron, J.-S. On the Exact Security of Full Domain Hash. In *Proceedings of the 20th Annual International Cryptology Conference on Advances in Cryptology* (London, UK, 2000), CRYPTO '00, pp. 229–235.

[10] Dept of Transportation. Dedicated Short Range Communications. `http://www.standards.its.dot.gov/Documents/advisories/dsrc_advisory.htm`. ITS Standards Advisories No.3.

[11] Hubbard, P. Urban congestion–why 'free' roads are costly. Treasury Department, Commonwealth of Australia.

[12] Klein, L. A. *Sensor Technologies and Data Requirements for ITS*. Artech House, 2001, ch. 7.

[13] Litman, T. London Congestion Pricing: Implications for Other Cities. `http://www.vtpi.org/london.pdf`, 2003.

[14] Lysyanskaya, A., Rivest, R. L., Sahai, A., and Wolf, S. Pseudonym Systems. In *Proceedings of the 6th Annual International Workshop on Selected Areas in Cryptography* (London, UK, 2000), SAC '99, pp. 184–199.

[15] Meiklejohn, S., Mowery, K., Checkoway, S.,

and Shacham, H. The Phantom Tollbooth: Privacy-Preserving Electronic Toll Collection in the Presence of Driver Collusion. In *Proceedings of USENIX Security 2011* (San Francisco, CA, August 2011).

[16] Newmarker, C. Toll Records Catch Unfaithful Spouses. `http://www.usatoday.com/tech/news/surveillance/2007-08-10-ezpass_N.htm"`, August 2007.

[17] Page, D., Smart, N. P., and Vercauteren, F. A comparison of MNT curves and supersingular curves. *Appl. Algebra Eng., Commun. Comput. 17* (October 2006), 384.

[18] Popa, R. A., Balakrishnan, H., and Blumberg, A. VPriv: Protecting Privacy in Location-Based Vehicular Services. In *18th USENIX Security Symposium* (Montreal, Canada, August 2009).

[19] Rieback, M. R., Crispo, B., and Tanenbaum, A. S. The Evolution of RFID Security. *IEEE Pervasive Computing 5* (January 2006), 62–69.