Cryptanalysis of the Privacy-Preserving Ride-Hailing Service TRACE

Deepak Kumaraswamy\textsuperscript{1}[0000–1111–2222–3333] and Srinivas Vivek\textsuperscript{2}[0000–0002–8426–0859]

\textsuperscript{1} National Institute of Technology Karnataka, India
deepekumaraswamy99@gmail.com
\textsuperscript{2} International Institute of Information Technology Bangalore, India
srinivas.vivek@iiitb.ac.in

Abstract. In a typical ride-hailing service, the service provider (RS) matches a customer (RC) with the closest vehicle (RV) registered to this service. Ride-hailing services have gained tremendous popularity over the past years, and several works have been proposed to ensure privacy of riders and drivers during ride-matching. TRACE is an efficient privacy-preserving ride-hailing service proposed by Wang et al. (IEEE Trans. Vehicular Technology 2018). TRACE uses masking along with other cryptographic techniques to ensure efficient and accurate ride-matching. RS computes a (secret) spatial division of a region into quadrants. The RS uses masked location information to match RCs and RVs within a quadrant without obtaining their exact locations, thus ensuring privacy. Additionally, an RC only gets to know location of the closest RV finally matched to it, and not of other responding RVs in the region.

In this work, we disprove the privacy claims in TRACE by showing the following: a) RCs and RVs can identify the secret spatial division maintained by RS (this reveals information about the density of RVs in the region and other potential trade secrets), and b) the RS can identify exact locations of RCs and RVs (this violates location privacy). Prior to exchanging encrypted messages in the TRACE protocol, each entity masks the plaintext message with a secret unknown to others. Our attack allows other entities to recover this plaintext from the masked value by exploiting shared randomness used across different messages, that eventually leads to a system of linear equations in the unknown plaintexts. This holds even when all the participating entities are honest-but-curious. We implement our attack and demonstrate its efficiency and high success rate. For the security parameters recommended for TRACE, an RV can recover the spatial division in less than a minute, and the RS can recover the location of an RV in less than a second on a commodity laptop.

Keywords: Location Privacy · Privacy-Preserving Protocols · Ride-Hailing Services · Cryptanalysis · Random Masking

© Springer Nature Switzerland AG 2021. The final published version is available at www.springerlink.com. DoI: 10.1007/978-3-030-92518-5
1 Introduction

Ride-hailing services such as Uber and Lyft have become a popular choice of transportation in the past decade [9]. By offering convenience and reliability to its customers, these services are well suited for intra-city commutes. A ride-hailing service usually consists of three entities: the ride-hailing server (RS), riders or customers (RCs) and drivers or vehicles (RVs). The RS is primarily responsible for hosting the ride-hailing service publicly. Drivers can register to this service and become identified as certified RVs. A customer who wishes to make use of this service can sign up as an RC and request for a ride. Depending on the pick-up and destination locations, the RS smartly forwards this ride request from RC to suitable RVs in the region. A list of nearby available RVs is revealed to the RC along with their reputations, who then makes a suitable choice.

However, revealing locations of RCs/RVs to other entities can have severe consequences. A pick-up location could correspond to the residential address of an RC, which can be used for stalking/kidnapping. There have also been instances when RVs registered to a particular ride-hailing service have been targeted by regular taxi drivers or targeted for theft [14,4]. Preserving privacy of sensitive users’ locations has become a primary concern in ride-hailing services. Generally, the RS is assumed to be honest-but-curious. This means that RS tries to learn as much information as possible without maliciously deviating from the ride-hailing protocol. Such a model is reasonable to assume since the RS wishes to preserve its reputation among the public. But it is still dangerous for the RS to learn locations of RCs and RVs, in case the RS later turns malicious or becomes a victim of cyberattacks [3,8].

In the past few years, there have been many works that focus on ensuring location privacy of RCs and RVs in the context of ride-hailing services. Section 5 contains an overview of recent papers in this area. These works use cryptographic primitives to hide sensitive location information from the RS, while trying to ensure efficiency and ride-matching accuracy.

In this paper, we focus on TRACE [16], proposed by Wang et. al. in 2018. TRACE is a privacy-preserving solution to ride-hailing services. Here, the RS first spatially divides each city into quadrants. RCs and RVs mask their sensitive location information using randomness and then forward it to RS. The RS then identifies the quadrant in which RCs and RVs lie, without finding out their exact locations. To ensure efficiency and accuracy, the ride request from an RC is forwarded only to RVs that are in the same quadrant as RC. The RC then makes a choice among RVs that lie in its vicinity to finalize ride establishment. Since the RS knows the distribution of RVs in different quadrants, it can periodically change its spatial division of the city to optimize bandwidth usage, reduce waiting time and improve accuracy.

TRACE uses masking with random secrets to prevent other entities of the protocol from learning the underlying message. At a high level, a large prime $p$ is chosen and the plaintext is multiplied with a random integer in $\mathbb{Z}_p$. These masked messages are encrypted using shared keys to prevent external eavesdroppers from
gaining any useful information. Since TRACE uses lightweight cryptographic
techniques and simple modular arithmetic, it is efficient in practice. The security
guarantees for TRACE state that RS cannot learn about the exact locations of
RCs and RVs apart from the quadrant they are in. Additionally, RCs and RVs
cannot learn about the secret spatial division maintained by RS, since this could
reveal the density of drivers across the city, among other proprietary information
and trade secrets of RS.

1.1 Our Contribution

We propose an attack on TRACE and disprove the above security claims by
showing that the RS can indeed retrieve the exact locations of all RCs and RVs.
Secondly, we show that RCs and RVs can learn the secret spatial division informa-
tion maintained by RS. These attacks constitute a total break of the privacy
objectives of TRACE. The underlying idea behind our attack is to eliminate
the (unknown) randomness shared across different messages when other entities
mask their location values. This allows one to efficiently obtain an overdeter-
dined system of linear (modular) equations in the unknown plaintext locations.
We stress that this attack is purely algebraic, and does not make any geometric
assumptions about the region. Our attack is efficient (runs in time quadratic in
the security parameters) and holds even when all entities are honest-but-curious.
For instance, with the recommended security parameters from [16], an RV can
recover the quadtree maintained by RS in under a minute (see Table 2) and the
RS can recover the exact location of an RV in under a second (see Table 3).

The rest of our paper is organized as follows. In Section 2, we describe relevant
steps of the TRACE protocol from [16]. The first attack in Section 3.1 describes
how RCs and RVs can recover the secret quadtree maintained by RS. The second
attack in Section 3.2 describes how the RS can recover exact locations of RCs
and RVs. We briefly discuss a modification to the TRACE protocol that prevents
only the first attack, and argue that the second attack (which is more severe
than the first) is hard to thwart. Algorithms 1 and 2 summarize the above two
attacks. Section 4 provides details about our experimental setup and evaluates
the efficiency and success rate of our attack in practice (refer Tables 2 and 3). Section 5 gives an overview of recent works in the area of privacy-preserving
ride-hailing services. We conclude our paper and provide remarks about future
work in Section 6.

2 Overview of TRACE

This section contains a high level overview of the TRACE protocol [16]. Details
that are not directly relevant to our attack will be omitted. For more information
the reader is referred to the original paper.
2.1 Preliminaries

A quadtree \( \{N_1, \ldots, N_m\} \) with \( m \) nodes is a data structure used to represent the partition of a 2-D space into quadrants and subquadrants. Each node \( N_i \) in the tree is associated with four \((x, y)\) coordinates denoting corners of the quadrant represented by that node. Every non-leaf node in the quadtree has four children denoting the division of that quadrant into four subquadrants. An example is presented in Figure 1.

![Fig. 1. Example of a quadtree](image)

Given a point \( P = (x, y) \) and a quadrant \( \{(x_j, y_j)\} \) with \( j = 1, \ldots, 4 \), we can easily check if \( P \) lies within the quadrant by doing the following [16, Section III]. For each \( j \) compute

\[
S_j = (xy_j + yx_j' + x_jy_j') - (xy_j' + yx_j + x_j'y_j) \quad (1)
\]

where \( j' = (j \mod 4) + 1 \). If all \( S_j \geq 0 \), then \( P \) lies within the quadrant, otherwise it does not. Given a quadtree, this idea can be extended to find the quadrant/node of the tree in which \( P \) lies. Starting at the root, among its four children, find that quadrant/node in which \( P \) lies; then recurse on its children until a leaf is encountered.

2.2 System Design and Security Goals

System Design. The three primary entities in the TRACE protocol are the ride-hailing server/service provider (RS), the customer/rider (RC) and the vehicle/driver (RV). All of the aforementioned entities are assumed to be honest-but-curious. This means that they wish to learn as much information as they can about the other entities without violating any protocol steps.

RS is mainly responsible for forwarding requests/responses between RCs and RVs. As part of the protocol, RS maintains a spatial division of the city into quadrants and uses it to identify regions in which RCs and RVs lie. It does so in such a way that RCs and RVs do not learn any information about the spatial division, while RS does not learn the exact locations of RCs and RVs. The RC can choose a pick-up point and send a ride-hailing request to RS, who then forwards it to the RVs that lie in close vicinity of RC. RVs submit their
masked location information to RS at regular intervals, allowing the RS to have an idea of distribution of RVs in the city. Depending on the density of RVs, RS can periodically optimize its space division to improve ride-matching accuracy.

**Threat Model.** We assume the same threat model that is considered in TRACE. All entities are assumed to be honest-but-curious, that is, they follow the protocol specification but may infer additional data from the observed transcripts. RS does not collude with RCs and RVs (to try and obtain information about customers), since it has an incentive to maintain high reputation.

**Security Goals.** It is essential to ensure that location information of RCs and RVs is not revealed to other entities. The spatial division maintained by RS should also be kept secret, as this could reveal information about density of drivers in a city and other proprietary information/trade secrets of RS. The authors of TRACE claim that the following security requirements are satisfied during the protocol execution.

**Claim 1** RS creates a quadtree $N$ containing information about spatial division of the city into quadrants, and masks it with a randomly chosen secret to compute $EN$. Given $EN$, RCs and RVs do not learn anything about $N$.

**Claim 2** RS can only learn the quadrants in which RCs lie. RS does not obtain any other information about the exact pick-up locations of RCs.

**Claim 3** RS can only learn the quadrants in which RVs lie. RS does not obtain any other information about the exact locations of RVs.

### 2.3 TRACE Protocol

This section describes the execution of the TRACE protocol. Figure 2 gives a summarized view of the messages exchanged between different entities. RS acts as a central entity for forwarding messages between RCs and RVs. It establishes shared keys with RCs and RVs through the Diffie-Hellman key exchange. All messages exchanged between RS and RCs, RVs are encrypted using a symmetric encryption scheme. The authentication of entities is ensured by signing these messages using the BLS signature scheme [1]. The notations used in the TRACE protocol and their descriptions are provided in Table 1.

For convenience, the remainder of this paper shall refer to subscripts $(\cdot)_{i,j}$ and $(\cdot)_{i,j,l}$ as simply $(\cdot)_{ij}$ and $(\cdot)_{ijl}$, respectively.

**Step 0.** RS publishes details about different system parameters (for example, the group and its generator used in the signature scheme, public key of RS, choice of symmetric encryption). RCs and RVs also establish their public keys. RS announces security parameters $k_1, k_2, k_3, k_4$. As we shall see subsequently, they specify the size of different randomness used when masking location information. Step 3 elaborates on the constraint that should exist among these four parameters to ensure correctness of the protocol.

RS chooses two large public primes $p$ and $\alpha$ (of size $k_1$ bits and $k_2$ bits, respectively) and a random secret $s \in \mathbb{Z}_p^*$ known only to itself.
Fig. 2. Overview of TRACE protocol
| Notation | Description |
|----------|-------------|
| RS | Ride-hailing server (service provider) |
| RC | Customer (rider) |
| RV | Vehicle (driver) |
| $k_1, k_2, k_3, k_4$ | Security parameters of TRACE |
| $N$ | Spatial division (quadtree) maintained by RS |
| $\alpha, p$ | Large primes chosen by RS |
| $\alpha', p'$ | Large primes chosen by RC |
| $x_{Nij}, y_{Nij}$ | Coordinates of $j$-th vertex in the $i$-th quadtree node $N_i$ |
| $s, a_{jh}$ | Random values used by RS when masking $N$ |
| $EN$ | Masked quadtree computed by RS |
| $(x_V, y_V)$ | Coordinates of RV |
| $r_{ij}$ | Random values chosen by RV when masking $(x_V, y_V)$ |
| $\pi(\cdot)$ | Random permutation chosen by RV |
| $A$ | Data aggregated by masking $(x_V, y_V)$ and $EN$ |
| $(x_{CP}, y_{CP})$ | Pick-up coordinates of RC |
| $S_{RC}$ | Square with $(x_{CP}, y_{CP})$ at its center |
| $R$ | Length of a side of $S_{RC}$ |
| $s', d_i$ | Random values used by RC when masking $(x_{CP}, y_{CP})$ |
| $C_1, \ldots, C_5$ | Data aggregated by masking $(x_{CP}, y_{CP})$ and $EN$ |

**Table 1. Description of the notations used in the TRACE protocol.**

**Step 1.** RS divides the two-dimensional space into squares or rectangles represented by a quadtree

$$N = \{N_1, N_2, \ldots, N_m\}$$

with $m$ nodes. The $i$-th quadrant $N_i$ has four corners $\{N_{ij} = (x_{Nij}, y_{Nij})\}$ where $j = 1, \ldots, 4$. RS wishes to learn the quadrant in which each RV lies without learning its exact location. To do this, RS sends a masked version of $N_i$ to RV. Concretely, RS chooses 24 random values $a_{jh}$ ($j = 1, \ldots, 4; h = 1, \ldots, 6$) of size $k_3$ bits each. For every vertex $N_{ij}$ of $N_i$, let $N_{ij'}$ be the vertex adjacent to it in the anticlockwise direction, i.e. $j' = (j \mod 4) + 1$. RS masks this vertex by
computing

\[ EN_{ij} = s(x_{Nij} \cdot \alpha + a_{i1}) \mod p , \]
\[ EN_{ij2} = s(y_{Nij} \cdot \alpha + a_{i2}) \mod p , \]
\[ EN_{ij3} = s(x_{Nij} \cdot \alpha + a_{i3}) \mod p , \]
\[ EN_{ij4} = s(y_{Nij} \cdot \alpha + a_{i4}) \mod p , \]
\[ EN_{ij5} = s(x_{Nij} \cdot y_{Nij} \cdot \alpha + a_{i5}) \mod p , \]
\[ EN_{ij6} = s(x_{Nij} \cdot y_{Nij} \cdot \alpha + a_{i6}) \mod p . \]

The values \( \alpha, p \) are public, whereas \( s, x_{Nij}, y_{Nij} \) are only known to RS. The masked coordinate is

\[ EN_{ij} = EN_{ij1} \parallel EN_{ij2} \parallel EN_{ij3} \parallel EN_{ij4} \parallel EN_{ij5} \parallel EN_{ij6} , \]

where \( \parallel \) denotes concatenation. Next, RS computes the masked quadrant

\[ EN_i = EN_{i1} \parallel EN_{i2} \parallel EN_{i3} \parallel EN_{i4} , \]

for \( i = 1, \ldots, m \), to get the masked quadtree

\[ EN = \{ EN_1, EN_2, \ldots, EN_m \} . \]

It then encrypts \( EN \) and forwards it to RV.

**Step 2.** RV decrypts this message and uses \( EN \) along with its own randomness to mask its location \((x_V, y_V)\). For \( i = 1, \ldots, m; j = 1 \ldots, 4 \), RV chooses a fresh random number \( r_{ij} \) (each \( k_4 \) bits long) and computes

\[ A_{ij1} = r_{ij} \cdot \alpha(x_V \cdot EN_{ij4} + y_V \cdot EN_{ij1} + EN_{ij6}) \mod p , \]
\[ A_{ij2} = r_{ij} \cdot \alpha(x_V \cdot EN_{ij2} + y_V \cdot EN_{ij3} + EN_{ij5}) \mod p , \]
\[ A_{ij} = A_{ij1} \parallel A_{ij2} . \]

RV chooses a random permutation \( \pi(\cdot) \) to reorder the \( j \)-indices for each \( A_i \). That is,

\[ A_i = A_{i\pi(1)} \parallel A_{i\pi(2)} \parallel A_{i\pi(3)} \parallel A_{i\pi(4)} , \]
\[ A = \{ A_1, \ldots, A_m \} . \]

The order within each \( A_{i\pi(j)} \) is still preserved, that is,

\[ A_{i\pi(j)} = A_{i\pi(j)1} \parallel A_{i\pi(j)2} . \]

RV encrypts \( A \) and forwards it to RS.

**Step 3.** RS obtains \( A \) that contains the masked location of each RV, and does the following computations to identify the quadrant/node \( N_i \) of the quadtree in which RV lies.
\[ B_{ij1} = s^{-1} \cdot A_{ij1} \mod p \]
\[ = s^{-1} \cdot r_{ij} \cdot \alpha(x_v \cdot E_{Nij4} + y_v \cdot E_{Nij1} + E_{Nij6}) \mod p \]
\[ = s^{-1} \cdot r_{ij} \cdot s[\alpha^2(x_v \cdot y_{Nij'} + y_v \cdot x_{Nij} + x_{Nij'} \cdot y_{Nij}) \]
\[ + \alpha(x_v \cdot a_j4 + y_v \cdot a_j1 + a_j6)] \mod p . \]
\[ B'_{ij1} = \frac{B_{ij1} - (B_{ij1} \mod \alpha^2)}{\alpha^2} \]
\[ = r_{ij}(x_v \cdot y_{Nij'} + y_v \cdot x_{Nij} + x_{Nij'} \cdot y_{Nij}) . \]

Similarly,
\[ B_{ij2} = s^{-1} \cdot A_{ij2} \mod p , \]
\[ B'_{ij2} = \frac{B_{ij2} - (B_{ij2} \mod \alpha^2)}{\alpha^2} \]
\[ = r_{ij}(x_v \cdot y_{Nij} + y_v \cdot x_{Nij'} + x_{Nij} \cdot y_{Nij'}) . \]

Next, RS computes the difference
\[ B_{ij} = B_{ij2} - B_{ij1} \]
\[ = r_{ij}[(x_v \cdot y_{Nij} + y_v \cdot x_{Nij'} + x_{Nij} \cdot y_{Nij'}) \]
\[ - (x_v \cdot y_{Nij'} + y_v \cdot x_{Nij} + x_{Nij'} \cdot y_{Nij})] . \]

Compare this to Equation (1). Since \( r_{ij} \) is always positive, RS can identify whether RV lies in \( N_i \) by checking if \( B_{ij} \) is positive for all \( j = 1, \ldots, 4 \). Using the method described in Section 2.1, RS can query the quadtree to identify the exact quadrant where RV lies.

Note that it was necessary to remove the modulus with respect to \( p \) when obtaining \( B'_{ij1} \) and \( B'_{ij2} \), otherwise those values would always be positive irrespective of whether RV was inside the quadrant \( N_i \) or not. To remove this modulus it is sufficient if the following is always true during the computation of \( B_{ij1} \) (a similar condition exists for \( B_{ij2} \)).
\[ r_{ij}[\alpha^2(x_v \cdot y_{Nij'} + y_v \cdot x_{Nij} + x_{Nij'} \cdot y_{Nij}) \]
\[ + \alpha(x_v \cdot a_j4 + y_v \cdot a_j1 + a_j6)] < p , \]
\[ r_{ij} \cdot \alpha(x_v \cdot a_j4 + y_v \cdot a_j1 + a_j6) < \alpha^2 , \]
\[ r_{ij} \cdot \alpha(x_v \cdot a_j2 + y_v \cdot a_j3 + a_j5) < \alpha^2 . \]

Let \( \langle \cdot \rangle \) denote the bit length of a non-negative integer. Recall that \( \langle p \rangle = k_1, \langle \alpha \rangle = k_2, \langle a_jh \rangle = k_3, \langle r_{ij} \rangle = k_4 \). To ensure the above conditions hold, the parameters are chosen such that
\[ k_4 + 2k_2 < k_1 , \]
\[ k_2 + k_3 < k_1 , \]
\[ k_3 + k_4 < k_2 \, . \]
Moreover, the size of location coordinates are assumed to be negligible compared to these security parameters. In [16], the above values are set as $k_1 = 512$, $k_2 = 160$, $k_3 = 75$, $k_4 = 75$.

**Step 4.** RC receives $EN$ from RS. Now the RC tries to mask its location with respect to the quadtree and send it to RS. Suppose the pick-up point of RC is $(x_{CP}, y_{CP})$. RC chooses a square $S_{RC}$ of side $2R$ (where $R$ is $\geq 1$ km) with this pickup point at its center. Let the vertices of this square be $\{(x_{S1}, y_{S1}), (x_{S2}, y_{S2}), (x_{S3}, y_{S3}), (x_{S4}, y_{S4})\}$. Recall that in Step 2, each RV masked its location $(x_V, y_V)$ with respect to $EN$ and computed $A$. RC also does an equivalent computation here; after receiving $EN$ from RS, it computes a masking for each of the four vertices of $S_{RC}$ to obtain $C = C_1 \parallel C_2 \parallel C_3 \parallel C_4$.

Next, RC chooses a public prime $p'$ of size $k_1$ bits, a public prime $\alpha'$ of $k_2$ bits, a secret $s' \in \mathbb{Z}_{p'}^*$ and 4 random values $d_i$ of $k_4$ bits each. It computes

\begin{align*}
D_1 &= s'(x_{CP} \cdot \alpha' + d_1) \mod p', \\
D_2 &= s'(y_{CP} \cdot \alpha' + d_2) \mod p', \\
D_3 &= s' \cdot d_3 \mod p', \\
D_4 &= s' \cdot d_4 \mod p', \\
D &= D_1 \parallel D_2 \parallel D_3 \parallel D_4, \\
E &= x_{CP}^2 + y_{CP}^2 - R^2.
\end{align*}

RC encrypts $C, D, E$ and sends it to RS.

**Step 5.** The goal here is to convey the masked location information from RC to RVs that are “nearby” to it. RS decrypts the message from RC to get $C, D, E$. Similar to Step 3, for each of $C_1, C_2, C_3, C_4$, RS obtains the quadrant in which the vertex represented by $C_i$ (i.e. $(x_{Si}, y_{Si})$) lies. With this RS knows the quadrants in which the corners of square $S_{RC}$ lies. RS can construct a region $CS_{RC}$ enclosing $S_{RC}$. From Step 3, RS also knows the quadrants in which each RV lies. RS encrypts $D, E$ and sends it to those RVs that lie in $CS_{RC}$ (call these RVs as SRVs).

**Step 6.** SRV receives $D, E$ from RS and tries to add in masked information about its own location $(x_{SV}, y_{SV})$ to these values. It chooses three random $r_i$’s of $k_4$ bits each and computes

\begin{align*}
F_1 &= x_{SV} \cdot \alpha' \cdot D_1 \mod p', \\
F_2 &= y_{SV} \cdot \alpha' \cdot D_2 \mod p', \\
F_3 &= r_3 \cdot D_3 \mod p', \\
F_4 &= r_2 \cdot D_4 \mod p', \\
F &= r_3(F_1 + F_2 + F_3 + F_4), \\
I &= r_3(x_{SV}^2 + y_{SV}^2 + E).
\end{align*}

SRV encrypts and sends $I, F$ to RC via RS.
**Step 7.** RC uses $I, F$ (that contain masked information of RC’s and SRVs’ locations) to check if that SRV is within distance $R$.

\[
J = s^{-1} \cdot F \mod p' = s^{-1} \cdot s' \cdot r_3 \alpha'^2 (x_{CP} \cdot x_{SV} + y_{CP} \cdot y_{SV}) + \alpha' (x_{SV} \cdot d_1 + y_{SV} \cdot d_2) + r_1 \cdot d_3 + r_2 \cdot d_4 \mod p',
\]
\[
J' = \frac{J - (J \mod \alpha'^2)}{\alpha'^2} = r_3 (x_{CP} \cdot x_{SV} + y_{CP} \cdot y_{SV}),
\]
\[
K = I - 2J' = r_3 [x_{CP}^2 + y_{CP}^2 + x_{SV}^2 + y_{SV}^2 - 2(x_{CP} \cdot x_{SV} + y_{CP} \cdot y_{SV}) - R^2]
\]
\[
= r_3 [(x_{CP} - x_{SV})^2 - (y_{CP} - y_{SV})^2 - R^2].
\]

When $K \leq 0$, the SRV is within the circle query range $C_{RC}$ of radius $R$ around $RC$. Call such SRVs as CRVs.

Once again (similar to Step 3) we need to eliminate the modulus with respect to $p'$ (otherwise $K$ would always be positive even if the SRV had distance $> R$). With the relationship imposed on the security parameters (Equation (2) in Step 3), the following condition holds and the modulus is removed.

\[
r_3 \alpha'^2 (x_{CP} \cdot x_{SV} + y_{CP} \cdot y_{SV}) + \alpha' (x_{SV} \cdot d_1 + y_{SV} \cdot d_2) + r_1 \cdot d_3 + r_2 \cdot d_4 < p',
\]
\[
r_5 \alpha' (x_{SV} \cdot d_1 + y_{SV} \cdot d_2) + r_1 \cdot d_3 + r_2 \cdot d_4 < \alpha'^2.
\]

**Step 8.** RC masks its take-off point $(x_{CT}, y_{CT})$ using $EN$ (similar to Step 2) to create $C_5$, and forwards $C_5$ along with the list of CRVs to RS. (The take-off point usually lies very close to the RC’s pick-up point from Step 4). Similar to Step 3, RS uses $C_5$ to identify the subregion in which the take-off point lies. RS chooses a random location $ATP$ in this subregion and forwards it to CRVs. Each CRV inspects $ATP$ to make a decision on whether to accept this ride-hailing request from RC. The CRVs who decide to accept send an “Accept Response” to RS. RS forwards the list of ready and available CRVs to RC. RC chooses a suitable CRV from this list, and this CRV is informed about the same by RS. Later, the RC and the chosen CRV proceed with ride establishment by negotiating a shared session key and by exchanging information such as location, phone number, reputation, etc.

## 3 Attack on TRACE

This section presents two attacks which (with high empirical probability) disprove the following privacy claims made about TRACE. First, in Section 3.1, we show that RCs and RVs can obtain the secret spatial division (quadtree)
information maintained by RS (violation of Claim 1). We also discuss a modification to the TRACE protocol, as a countermeasure for this attack. Secondly, in Section 3.2, we show how the RS can identify exact locations of all RCs and RVs (violation of Claims 2, 3). We also briefly argue why this attack is not straightforward to thwart. In both attacks, the entities recover location coordinates modulo prime $p$. This is same as recovering the actual integer values since $p$ is a very large prime and the coordinate values are negligibly small compared to $p$.

Steps from the TRACE protocol described in Section 2.3 will be referred as and when needed. In Section 4, we shall experimentally evaluate the success probability of our attacks.

### 3.1 RCs, RVs obtain Quadtree

After an RV receives the masked quadtree $EN$ computed by RS (Step 2), we show how it can recover all underlying vertices $x_{Ni}, y_{Ni}$ of the quadtree’s nodes. This same principle allows an RC to obtain information about the quadtree as well (recall that each RC receives $EN_1, \ldots, EN_m$).

**Intuition.** Intuitively, our attack works as follows. Each quadtree node $N_i$ is masked by the RS using random values $s, a, a_{jh}$, resulting in $EN_i$. When an RV receives $EN_1, \ldots, EN_m$, it knows $p, a$ but does not know $s, a_{jh}$. For a single $EN_i$, the number of equations involved is $4 \times 6 = 24$ (since there is one equation for each $EN_{ijh}, j = 1, \ldots, 4; h = 1, \ldots, 6$). The number of unknowns involved in $EN_i$ is $1 + 24 + 8 = 33$ ($s, a_{jh}$’s and quadrant vertices $x_{Ni}, y_{Ni}$, $j = 1, \ldots, 4; h = 1, \ldots, 6$). A key observation is that if one considers $EN_i$ along with a different $EN_i'$, the number of equations is $24 + 24 = 48$. However the number of unknowns involved is $1 + 24 + 8 + 8 = 41$ ($s, a_{jh}$’s and quadrant vertices $x_{Ni}, y_{Ni}, x_{Ni'}, y_{Ni'}$, where $j = 1, \ldots, 4; h = 1, \ldots, 6$). That is, considering an additional $EN_i'$ gives 24 new equations but introduces only 8 new variables. This would allow RV to solve this system of modular equations and obtain the secrets $s$ along with quadrant vertices of $N_i$ and $N_i'$.

**Formal attack.** Without loss of generality, we show how an RV can recover vertices of quadrants $N_1, N_2$ when given $EN_1, EN_2$ (i.e. $i = i' = 2$). The first task is to eliminate the unknown randomness $a_{jh}, j = 1, \ldots, 4; h = 1, \ldots, 6$. This can be done by subtracting $EN_{2jh}$ from $EN_{1jh}$. For $h = 1, \ldots, 6$, we get the following equations.

\[
\begin{align*}
EN_{1j1} - EN_{2j1} &= s \alpha (x_{N1j} - x_{N2j}) \mod p, \\
EN_{1j2} - EN_{2j2} &= s \alpha (y_{N1j} - y_{N2j}) \mod p, \\
EN_{1j3} - EN_{2j3} &= s \alpha (x_{N1j'} - x_{N2j'}) \mod p, \\
EN_{1j4} - EN_{2j4} &= s \alpha (y_{N1j'} - y_{N2j'}) \mod p, \\
EN_{1j5} - EN_{2j5} &= s \alpha (x_{N1j} y_{N1j'} - x_{N2j} y_{N2j'}) \mod p, \\
EN_{1j6} - EN_{2j6} &= s \alpha (x_{N1j'} y_{N1j} - x_{N2j'} y_{N2j}) \mod p.
\end{align*}
\]
Here \( j' = (j \mod 4) + 1 \). The parameters \( s, \alpha \) are unknown to RV along with the 16 variables \( x_{N1j}, y_{N1j}, x_{N2j}, y_{N2j}, j = 1, \ldots, 4 \). RV can obtain linear (modular) equations in these variables by eliminating \( s, \alpha \) as follows.

Compare (3) \( \times y_{N1j'} + (6) \times x_{N2j} \) and (7):
\[
(EN_{1j1} - EN_{2j1}) \times y_{N1j'} + (EN_{1j4} - EN_{2j4}) \times x_{N2j}
= s\alpha(x_{N1j}y_{N1j'} - x_{N2j}y_{N1j'} + y_{N1j'}x_{N2j} - y_{N2j}'x_{N2j})
= s\alpha(x_{N1j}y_{N1j'} - x_{N2j}y_{N2j'})
= (EN_{1j5} - EN_{2j5}) \mod p .
\] (9)

Compare (3) \( \times y_{N2j'} + (6) \times x_{N1j} \) and (7):
\[
(EN_{1j1} - EN_{2j1}) \times y_{N2j'} + (EN_{1j4} - EN_{2j4}) \times x_{N1j}
= s\alpha(x_{N1j}y_{N2j'} - x_{N2j}y_{N2j'} + y_{N1j'}x_{N1j} - y_{N2j'}x_{N1j})
= s\alpha(x_{N1j}y_{N1j'} - x_{N2j}y_{N2j'})
= (EN_{1j5} - EN_{2j5}) \mod p .
\] (10)

Compare (4) \( \times x_{N1j'} + (5) \times y_{N2j} \) and (8):
\[
(EN_{1j2} - EN_{2j2}) \times x_{N1j'} + (EN_{1j3} - EN_{2j3}) \times y_{N2j}
= s\alpha(y_{N1j}x_{N1j'} - y_{N2j}x_{N1j'} + x_{N1j'}y_{N2j} - x_{N2j}y_{N2j})
= s\alpha(x_{N1j'}y_{N1j} - x_{N2j}y_{N2j})
= (EN_{1j6} - EN_{2j6}) \mod p .
\] (11)

Compare (4) \( \times x_{N2j'} + (5) \times y_{N1j} \) and (8):
\[
(EN_{1j2} - EN_{2j2}) \times x_{N2j'} + (EN_{1j3} - EN_{2j3}) \times y_{N1j}
= s\alpha(y_{N1j}x_{N2j'} - y_{N2j}x_{N2j'} + x_{N1j'}y_{N1j} - x_{N2j}y_{N1j})
= s\alpha(x_{N1j'}y_{N1j} - x_{N2j}y_{N2j})
= (EN_{1j6} - EN_{2j6}) \mod p .
\] (12)

Compare (3) and (4):
\[
(EN_{1j1} - EN_{2j1}) \times s\alpha(y_{N1j} - y_{N2j})
= (EN_{1j2} - EN_{2j2}) \times s\alpha(x_{N1j} - x_{N2j}) \mod p ,
\Rightarrow (EN_{1j2} - EN_{2j2})(x_{N1j} - x_{N2j})
- (EN_{1j1} - EN_{2j1})(y_{N1j} - y_{N2j}) = 0 \mod p .
\] (13)

Similarly, compare (4) and (5), and (5) and (6):
\[
(EN_{1j3} - EN_{2j3})(y_{N1j} - y_{N2j})
- (EN_{1j2} - EN_{2j2})(x_{N1j'} - x_{N2j'}) = 0 \mod p ,
(EN_{1j4} - EN_{2j4})(x_{N1j'} - x_{N2j'})
- (EN_{1j3} - EN_{2j3})(y_{N1j'} - y_{N2j'}) = 0 \mod p .
\] (14)
Consider Equations (9)—(15) for all $j = 1, \ldots, 4$: $j' = (j \mod 4) + 1$. There are 28 linear (modular) equations in the 16 unknowns $(x_{N1j}, y_{N1j}), (x_{N2j}, y_{N2j})$: $j = 1, \ldots, 4$. This can be treated as a linear system of equations with elements from the field $\mathbb{Z}_p$, and standard techniques from linear algebra such as Gaussian Elimination can be applied to find solutions for $X$ in $\mathbb{Z}_p$.

**Existence of a unique solution** Suppose we represent Equations (9)—(15) using matrix notation as $PX = Q$, where $\dim(P) = 28 \times 16$, $\dim(X) = 16 \times 1$, $\dim(Q) = 28 \times 1$, and vector $X$ represents the 16 unknown quadrant vertices of $N_1, N_2$. We observed that $\text{rank}(P) \leq 13 < 16$, and the RV cannot obtain unique solutions for $X$ from this system.

Hence we propose a modification to our attack such that $\text{rank}(P)$ equals the number of unknowns. Previously, considering only $N_1, N_2$ gave us 28 equations and $8 \times 2 = 16$ unknowns. If we instead consider $N_1, N_2, N_3$ and take $\binom{4}{2} = 3$ pairwise combinations, we end up with $28 \times 3 = 84$ equations and $8 \times 3 = 24$ unknowns (which is slightly better). But we observed that in some cases, the resulting $84 \times 24$ matrix $P$ had rank $23 < 24$. Next, considering $N_1, N_2, N_3, N_4$ and taking $\binom{4}{3} = 6$ pairwise combinations gives us $28 \times 6 = 168$ equations and $8 \times 4 = 32$ unknowns. We observed (from experiments described in Section 4) that the corresponding $168 \times 32$ matrix $P$ always had rank 32, and an RV can therefore solve this system to get the unique values (in $\mathbb{Z}_p$) of quadrant vertices for $N_1, \ldots, N_4$. One can proceed further and consider more $N_i$, but that would be redundant since rank already equals the number of unknowns.

We now formalize the above idea. Let the linear system defined by Equations (9)—(15) (for vertices of $N_1, N_2$) be denoted by

$$
\begin{bmatrix}
P_{N1N2} & P_{N2N1}
\end{bmatrix}
\begin{bmatrix}
X_{N1} \\
X_{N2}
\end{bmatrix} = 
\begin{bmatrix}
Q_{N1N2}
\end{bmatrix}.
$$

Here $P_{N1N2}, X_{N1}$ and $P_{N2N1}, X_{N2}$ are submatrices corresponding to unknown vertices of $N_1$ and $N_2$, respectively. Note that $\dim(P_{N1N2}) = \dim(P_{N2N1}) = 28 \times 8$, $\dim(X_{N1}) = \dim(X_{N2}) = 8 \times 1$, $\dim(Q_{N1N2}) = 28 \times 1$. In the same manner, take all $\binom{4}{2} = 6$ pairwise combinations $N_i, N_j^\prime; 1 \leq i < i' \leq 4$ from $N_1, N_2, N_3, N_4$ and compute $P_{N_iN_i^\prime}, X_{N_i}, P_{N_i^\prime N_i}, X_{N_i^\prime}, Q_{N_iN_i^\prime}$. Define a linear system that considers all the above systems simultaneously.

$$
PX = Q,
$$

$$
\begin{bmatrix}
P_{N1N2} & P_{N2N1} & \cdots & P_{N3N4} & P_{N4N3}
\end{bmatrix}
\begin{bmatrix}
X_{N1} \\
X_{N2} \\
\vdots \\
X_{N4}
\end{bmatrix} = 
\begin{bmatrix}
Q_{N1N2} \\
Q_{N1N3} \\
\vdots \\
Q_{N3N4}
\end{bmatrix}.
$$

Here $0$ denotes the zero matrix of dimension $28 \times 8$, $\dim(P) = 168 \times 32$, $\dim(X) = 32 \times 1$, $\dim(Q) = 168 \times 1$ and $\text{rank}(P)$ is experimentally observed to be 32. The
Algorithm 1: RV recovers quadtree

| Input    | Size of quadtree \( m \), masked quadtree \( EN = (EN_1, \ldots, EN_m) \) |
|----------|-------------------------------------------------------------------|
| Output   | Underlying quadrant vertices \( N = (N_1, \ldots, N_m) \) |

Procedure Recover Quadtree(\( m \), \( EN \)):

- while \( \text{size}(EN) > 0 \) do
  - Pick four random entries \( EN_a, EN_b, EN_c, EN_d \) and delete them from \( EN \)
  - for each of the \( \binom{4}{2} \) pairwise combinations \( (i, i') \) from \( a, b, c, d \) do
    - Obtain a linear system in the unknown vertices of \( N_i, N_i' \) using equations similar to (9)–(15)
    - Let the corresponding matrices be \( P_{NiNi'}, X_{Ni}, P_{Ni'Ni'}, X_{Ni'} \), \( Q_{NiNi'} \) similar to (16)
  - end
  - Using the above matrices, define the system \( PX = Q \) similar to (17)
  - Solve this system to obtain quadrant vertices corresponding to \( N_a, N_b, N_c, N_d \)
- end

Output: \( (N_1, \ldots, N_m) \)

RV can solve this system to obtain unique solutions for \( X \) (i.e. quadrant vertices of \( N_1, \ldots, N_4 \)) in \( \mathbb{Z}_p \).

Note that there is no restriction here to use equations for the first four quadrants \( N_1, \ldots, N_4 \). The RV can consider equations corresponding to any four distinct \( N_i \) and find their underlying vertices. The above steps are repeated for other quadrants as well, until all of them are recovered. We summarize the attack in Algorithm 1. The same idea also allows an RC to recover the quadtree, when it receives \( EN \) from RS.

We remark that this attack is purely algebraic and does not make any assumptions about geometry of the region. The same attack would still work even if quadrants in the spatial division were not restricted to rectangles/squares.

**Complexity.** The linear system of equations represented by \( PX = Q \), where \( \dim(P) = 168 \times 32, \dim(X) = 32 \times 1, \dim(Q) = 168 \times 1 \), and all operations are in the field \( \mathbb{Z}_p \), can be solved in time \( O((\log p)^2) = O(k^2) \) [2]. We need to repeatedly solve such a system \( \lceil m/4 \rceil \) times to recover vertices of all \( m \) quadrants. The total asymptotic complexity of this attack is \( O(k^2 m) \). Our attack is efficient in practice, and Table 2 shows the average time taken to recover quadrant vertices for varying tree sizes and security parameters.

**Remark.** The aforementioned attack mainly relies on the fact that TRACE uses the same set of 24 random values \( a_{jh}, j = 1, \ldots, 4; h = 1, \ldots, 6 \) throughout all \( EN_i \), \( i = 1, \ldots, m \) (refer to Step 1 of the TRACE protocol in Section 2.3). However, upon careful observation, one can see that the correctness of the TRACE protocol would still hold if different set of values of \( a_{jh} \) were used for each \( EN_i \). That is, sample \( 24 \times m \) independent random values \( a_{ijh}, i = 1, \ldots, m; j = 1, \ldots, 4; h = 1, \ldots, 6 \) and mask \( EN_{ijh} \) with \( a_{ijh} \). In the TRACE protocol, these random values are involved only when computing \( B_{ij1} \).
and $B_{ij2}$ (Step 3). Correctness still holds since these values cancel each other out when computing $B_{ij1} - (B_{ij1} \mod \alpha^2)$.

Therefore, one can modify the TRACE protocol by using a new random $a_{ijh}$ each time when computing $EN_{ijh}$. This is a countermeasure to prevent RCs and RVs from obtaining the secret quadtree because, $EN_{ijh}$ is masked by fresh randomness each time and no information can be obtained about $(x_{Nij}, y_{Nij})$ given the $EN_i$ values (similar to a one-time pad). This simple observation leads to the following lemma.

**Lemma 1.** The above modification to TRACE provides information-theoretic security against any passive adversary who wishes to obtain additional information about the quadtree maintained by RS.

However, as we shall see in Section 3.2, this modification does not prevent the RS in obtaining locations of RCs and RVs. We will also later see that a similar countermeasure does not exist for the latter attack. Trying to use fresh randomness there will violate the correctness of the protocol.

### 3.2 RS obtains locations of RCs, RVs

**RS finds location of RVs.** In Step 3, RS receives $A_i = A_{i\pi(1)} || A_{i\pi(2)} || A_{i\pi(3)} || A_{i\pi(4)}$, $i = 1, \ldots, m$, from each RV, that contains masked information about $(x_V, y_V)$. RS knows $EN_{ij}$ but does not know $r_{ij}$ and the random permutation $\pi$ used on the four $A_{ij}$ values. Since there can only be 24 possible choices for $\pi$, the RS can enumerate all of them to try and find $\pi$.

For each $i$, RS initializes an empty set $S_i$. For each choice of permutation $\rho$ (among the set of all permutations on four elements), RS permutes the four components of $A_i$ according to $\rho$. That is, RS computes

$$A'_i = A_{i\rho(\pi(1))} || A_{i\rho(\pi(2))} || A_{i\rho(\pi(3))} || A_{i\rho(\pi(4))} = A'_{i1} || A'_{i2} || A'_{i3} || A'_{i4}. \quad (18)$$

$A'_i$ corresponds to the original value $A_1 || A_2 || A_3 || A_4$ computed by RV only when $\rho = \pi^{-1}$. To see if the current choice $\rho$ equals $\pi^{-1}$, RS can do the following: eliminate $r_{ij}, \alpha$ from $A'_{ij1}$ and $A'_{ij2}$ to get a linear equation in the unknowns $x_V, y_V$:

$$A'_{ij1}(x_V \cdot EN_{ij2} + y_V \cdot EN_{ij3} + EN_{ij5})$$

$$= A'_{ij2}(x_V \cdot EN_{ij4} + y_V \cdot EN_{ij1} + EN_{ij6}) \mod p. \quad (19)$$

In this way, RS can obtain four linear equations for $j = 1, \ldots, 4$, in the two unknowns $x_V, y_V$. Two of these equations can be used to solve and find $x_V, y_V$ (if a solution does not exist, move to the next choice of $\rho$). The remaining two equations can be used to check if the values of $x_V, y_V$ previously obtained are consistent. If so, then with high probability RS can infer that $\rho = \pi^{-1}$; add this solution to set $S_i$. If it is not consistent discard it and check the next permutation.
Algorithm 2: RS recovers location of an RV

**Input**: \((A_1, \ldots, A_m)\) representing masked information about an RV’s location

**Output**: RV’s location \((x_V, y_V)\)

**Procedure** Recover Location\((A_1, \ldots, A_m)\):

```plaintext
for \(i = 1, \ldots, m\) do
  \(S_i = \emptyset\)
  \(\mathcal{P} = \text{set of all permutations on 4 elements}\)
  for \(\rho \in \mathcal{P}\) do
    /* Permute the 4 components of \(A_i\) using \(\rho\) */
    Compute \(A'_i\) according to (18)
    Obtain a linear system in the unknowns \(x_V, y_V\) by substituting
    \(j = 1, 2\) in (19)
    if this system does not have a unique solution then
      continue
    end
    /* Solve this system and check if the unique solution \((x, y)\) also
    satisfies the two equations obtained when substituting \(j = 3, 4\) in
    (19) */
    if \((x, y)\) satisfies the consistency check then
      \(S_i = S_i \cup \{(x, y)\}\)
    end
    /* Note that \(\forall i, (x_V, y_V) \in S_i\) */
  end
  /* With high probability, we have \(|\cap_{i=1}^{m} S_i| = 1\) */
  \(\{(x_V, y_V)\} = \cap_{i=1}^{m} S_i\)
end
**Output**: \((x_V, y_V)\)
```

choice for \(\rho\). Note that \(S_i\) always contains the original \(x_V, y_V\) chosen by RV since this solution satisfies the consistency checks when \(\rho = \pi^{-1}\). In rare cases it could be possible that a false positive also passes these consistency checks for a different \(\rho\) and is added to \(S_i\).

The above procedure is discussed only for one value of \(i\). There are \(m\) such \(A_i\)’s received by RS (in general \(30 \leq m \leq 100\) \([16, \text{Section VI}]\)), and the original \(x_V, y_V\) is present in each \(S_i\). Moreover, it is highly unlikely that the same false positive appears in every \(S_i\). Therefore, it is very likely that there is only one common element present in all \(S_i\) (this probability increases with \(m\)), and that would be the required location of RV. Once again, there are no assumptions made regarding geometry of the spatial region. We summarize the attack in Algorithm 2.

**Complexity.** For each \(i = 1, \ldots, m\), RS enumerates over all 24 possible permutations. In each choice of permutation, RS solves a system of equations in two variables (with all elements being in \(\mathbb{Z}_p\)) and checks for consistency with two other equations to finally obtain the set \(S_i\). The size of each \(S_i\) is at most 24. RS later computes the intersection of all \(S_i\) to determine the RV’s coordinates. All
these operations can be done in time $O(k^2 m)$. Table 3 shows the average time taken to recover $(x_V, y_V)$ for varying tree size and security parameters.

**RS finds location of RCs.** In Step 5, RS receives $C_1 || C_2 || C_3 || C_4$ from RC. Recall that RC chooses a square $S_{RC}$ of side $2R$ with its pick-up location at the center. Each $C_j$ corresponds to one vertex of that square, masked in a manner similar to how RV masked its location as $A$ (refer to Step 2). Since we just saw an attack where RS can recover the original underlying location when given such a masking, RS can obtain the 4 vertices corresponding to $S_{RC}$. The center of this square directly gives the pick-up location of RC.

RS can also find the take-off location of RC. In Step 8, RS receives $C_5$ from RC, which is a masking of RC’s take-off location using $EN$ similar to what we have seen in Step 2. Using the same attack as for RV, RS can directly recover RCs location from $C_5$. Note that, in practice, the pick-up and the take-off points are quite close.

**Remark.** In Section 3.1 we saw that using fresh randomness for each encrypted quadtree term $EN_i$ did not violate correctness of the protocol. If we try to apply the same argument here, then in Step 2 of the TRACE protocol, $A_{ij1}$ and $A_{ij2}$ have to be masked with different (and fresh) randomness, say $r_{ij1}$ and $r_{ij2}$, respectively (currently, they are both masked by the same $r_{ij}$). But in Step 3 of the TRACE protocol, this would mean $B'_{ij1}$ is masked with $r_{ij1}$, and $B'_{ij2}$ is masked with $r_{ij2}$. Hence $B_{ij} = B'_{ij1} - B'_{ij2}$ would not have a common factor $r_{ij}$, and one cannot infer whether the RV lies inside the quadrant $N_i$ just by checking the sign of $B_{ij}$. Therefore, this approach will violate the correctness of the TRACE protocol, and we believe that other countermeasures for this attack are not straightforward to come up with.

### 4 Experimental Results

In this section, we discuss the experimental setup and other implementation aspects of the attacks mentioned in Section 3. Our experiments were implemented using SageMath [13] and run on an Intel Core i5-8250U CPU @ 1.60 GHz with 8 GB RAM running Ubuntu 20.04 LTS.

#### 4.1 Setup

The TRACE paper [16] states that setting $(k_1, k_2, k_3, k_4) = (512, 160, 75, 75)$ should be sufficient to ensure that Claims 1, 2, 3 hold. We also initialize these security parameters with the same values. In addition, we demonstrate the robustness of our attack by performing another set of experiments with larger values $(2048, 1000, 400, 400)$ satisfying Equation (2). Note that our attack is clearly independent of the security of encryption schemes/digital signatures used in TRACE.

---

3The implementation can be accessed at https://github.com/deepakkavoor/rhs-attack/tree/trace-attack.
The implementation of TRACE protocol from [16] does not give any reference to the dataset that was used to create spatial divisions. So, we simulate the creation of an arbitrary quadtree by first choosing an outermost rectangular quadrant, followed by picking a random center and dividing it into four sub-quadrants. We repeat this for the smaller quadrants until the number of nodes in the tree is $m$. The TRACE implementation in [16] varies $m$ between 28 and 84; we set $m = 50$ and $m = 100$ in our experiments. The attack indeed works for any value of $m$ (recall $m \geq 4$) and its success probability increases with $m$.

Integer modular arithmetic is used in all computations. Since the sizes of location coordinates are negligible compared to the security parameters $k_1, \ldots, k_4$, the vertices of the outermost quadrant are randomly chosen in the range $[0, 2^{20} - 1]$ (for $(k_1, k_2, k_3, k_4) = (512, 160, 75, 75)$) and in $[0, 2^{50} - 1]$ (for $(k_1, k_2, k_3, k_4) = (2048, 1000, 400, 400)$).

4.2 RCs, RVs recover Quadtree

RS computes the encrypted quadtree $EN$ and sends it to an RV as described in Section 2.3. Next, RV carries out the attack described in Section 3.1. We perform 20 iterations of this attack, and in each iteration, the RS generates a fresh random quadtree (as described in Section 4.1) and computes $EN$ accordingly. We observed that in all iterations, RV was able to recover the exact values of all quadrant vertices every time. We repeat the same for different choices of $m$ and security parameters, and tabulate the average time taken to recover the quadtree in Table 2. Since the same attack allows an RC to recover the quadtree, similar experimental statistics can be expected in this case.

| Security parameters $(k_1, k_2, k_3, k_4)$ | Size of quadtree $m$ | Time taken (in seconds) for an RV to recover quadtree, averaged over 30 iterations. |
|-------------------------------------------|---------------------|----------------------------------------------------------------------------------|
|                                          | 50                  | 55.686                                                                           |
|                                          | 100                 | 108.566                                                                          |
| (512, 160, 75, 75)                       |                     |                                                                                  |
| (2048, 1000, 400, 400)                   |                     |                                                                                  |
|                                          | 50                  | 2341.836                                                                         |
|                                          | 100                 | 4771.549                                                                         |

Table 2.

4.3 RS recovers locations of RCs and RVs

The location $(x_V, y_V)$ of an RV is randomly chosen within the outermost quadrant. We simulate the exchange of messages between RS and this particular RV, following the steps of TRACE protocol (Section 2.3). Next, RS carries out the attack described in Section 3.2. We perform 30 iterations of the attack with freshly generated (random) values for quadtree and $(x_V, y_V)$ in each iteration.
| Security parameters       | Size of quadtree m |
|--------------------------|-------------------|
| \((k_1, k_2, k_3, k_4)\) | 50                |
| \((512, 160, 75, 75)\)  | 0.206             |
| \((2048, 1000, 400, 400)\) | 7.461             |

Table 3. Time taken (in seconds) for RS to recover an RV’s location, averaged over 30 iterations.

We observed that in all iterations, RS was able to recover the exact location of the RV. That is, \(|\cap_{i=1}^m S_i|\) was exactly 1, and the recovered coordinate was same as the RV’s location in all iterations (refer Algorithm 2). We repeat the same for different choices of \(m\) and security parameters, and tabulate the average time taken to recover RV’s location in Table 3.

The attack to recover an RC’s location is exactly the same as that for an RV. Since we assume the distribution of RC’s location to be random as well, the same statistics also hold true when RS recovers the location of an RC.

5 Related Work

We briefly mention the prior works in privacy-preserving ride-hailing services. Since these works use fundamentally different ideas (such as homomorphic encryption, garbled circuits) compared to TRACE (which relies on random masking), our attack does not directly apply to these works.

PrivateRide by Pham et al. [11] is the first work that provides a practical solution towards privacy in ride-hailing systems. The locations and identities of riders are hidden using cloaked regions and anonymous credentials. They use efficient cryptographic primitives to ensure privacy of sensitive information. ORide by Pham et al. [10] offers accountability guarantees and secure payments along with privacy of riders and drivers. They use homomorphic encryption to compute the Euclidean distance and identify the closest driver in a zone. [5] proposes a modification to ORide to ensure location privacy of responding drivers in the region with respect to a rider.

Zhao et al. [20] conduct a study on leakage of sensitive data in ride-hailing services. They analyze APIs in non-privacy preserving apps provided to drivers by Uber and Lyft.

pRide by Luo et al. [7] proposes a privacy-preserving solution involving two non-colluding servers, one of them being the RS and the other a third-party Crypto Provider (CP). They use road network embedding in a higher dimension to approximate shortest distance over road networks. The homomorphically computed (approximate) distances are compared using a garbled circuit. Their scheme provides higher ride-matching accuracy than ORide while being computationally efficient. lpRide by Yu et al. [19] improves upon pRide by eliminating the need for a second Crypto Provider. They use a modified version of Paillier
cryptosystem for encrypting locations of riders and drivers. However, [15] proposed an attack on the modified Paillier scheme used in lpRide, allowing the service provider to recover locations of all riders and drivers in the region.

EPRide by Yu et al. [18] uses an efficient approach to compute the exact shortest road distance using road network hypercube embedding. They use somewhat homomorphic encryption over packet ciphertexts to achieve high ride-matching accuracy and efficiency, reporting significant improvements over ORide and pRide. Xie et. al. [17] improve upon pRide by combining the idea of road network embedding with cryptographic constructs such as Property-preserving Hash. They eliminate the need for a trusted third-party server to compute shortest distances.

Lu et al. [6] proposed a protocol for Privacy-Preserving Scalar Product (PP-SP) in 2013, which allows two parties \( P_0 \) and \( P_1 \) (having input vectors \( \overrightarrow{a} \) and \( \overrightarrow{b} \), respectively) to jointly compute the scalar product \( \overrightarrow{a} \cdot \overrightarrow{b} \) such that no information about \( P_i \)'s input is revealed to \( P_{1-i} \) (other than what is revealed by the output itself), for \( i \in \{0, 1\} \). Their protocol was claimed to achieve information-theoretic security using random masking, and does not make use of any computational assumptions. However, in 2019, [12] proposed an attack on the PP-SP protocol of Lu et al. and showed that it is impossible to construct a PP-SP protocol without the use of computational hardness assumptions. These attacks are based on constructing distinguishers that leak additional information about the other party’s secrets than what the output should reveal. While the TRACE protocol is motivated by the designs of the PP-SP protocols of Lu et al., we would like to stress that the application context, i.e., privacy-preserving ride-hailing services, is different in our setting, and hence the privacy requirements differ too. The main goal of our attacks on the TRACE protocol is the complete recovery of secret locations rather than just distinguishing them from uniform random values, and hence the attack techniques are also different. Note that the anonymity of users’ locations is the main requirement for a PP-RHS, and not just indistinguishability from uniform random values. Hence, the attack in [12] does not necessarily imply our results, though it certainly provides the motivation for a deeper investigation such as our work.

Also, the impossibility result of [12] does not necessarily imply that a PP-RHS cannot be constructed without computational hardness assumptions. For instance, in Section 3.1, we showed that our modification to the TRACE protocol, where fresh random values are used for each invocation, prevents RCs and RVs from obtaining the secret quadtree (this is based on an information-theoretic argument similar to that of a one-time pad). Hence, impossibility results for the PP-SP setting do not necessarily translate to the PP-RHS setting.

6 Conclusion and Future Work

In this work we proposed an attack on the privacy-preserving ride-hailing service TRACE. We disproved several privacy claims about TRACE in an honest-but-curious setting. We showed how riders (RCs) and drivers (RVs) can recover the
secret spatial division information maintained by the ride-hailing server (RS). We also showed how the RS can recover the exact locations of RCs and RVs. We implemented our attack and evaluated the success probability for different security parameters. In the future, it would be interesting to propose a modified protocol for TRACE in which all the aforementioned privacy claims hold.

Acknowledgements. This work was partially funded by the Infosys Foundation Career Development Chair Professorship grant for Srinivas Vivek.

References

1. Boneh, D., Lynn, B., Shacham, H.: Short signatures from the weil pairing. In: Boyd, C. (ed.) Advances in Cryptology — ASIACRYPT 2001. pp. 514-532. Springer Berlin Heidelberg, Berlin, Heidelberg (2001)
2. Eberly, W., Giesbrecht, M., Giorgi, P., Storjohann, A., Villard, G.: Solving sparse integer linear systems. CoRR abs/cs/0603082 (2006), http://arxiv.org/abs/cs/0603082
3. EconomicTimes: Bengaluru techie arrested for data theft from Aadhaar website. https://economictimes.indiatimes.com/small-biz/security-tech/security/ola-employee-arrested-for-data-theft-from-aadhaar-website/articleshow/5990079.cms?from=mdr (2017), retrieved: June 17, 2021
4. Hurriyet Daily News: Istanbul taxi drivers hunt down, beat up Uber drivers as tensions rise. https://www.hurriyetturkey.com/istanbul-taxi-drivers-hunt-down-beat-up-uber-drivers-as-tensions-rise-128443 (2018), retrieved: June 17, 2021
5. Kumaraswamy, D., Murthy, S., Vivek, S.: Revisiting driver anonymity in oride. CoRR abs/2101.06419 (2021), https://arxiv.org/abs/2101.06419, to appear in SAC 2021.
6. Lu, R., Lin, X., Shen, X.: Spoc: A secure and privacy-preserving opportunistic computing framework for mobile-healthcare emergency. IEEE Transactions on Parallel and Distributed Systems 24(3), 614-624 (2013). https://doi.org/10.1109/TPDS.2012.146
7. Luo, Y., Jia, X., Fu, S., Xu, M.: pRide: Privacy-Preserving Ride Matching Over Road Networks for Online Ride-Hailing Service. IEEE Trans. Information Forensics and Security 14(7), 1791-1802 (2019). https://doi.org/10.1109/TIFS.2018.2885282, https://doi.org/10.1109/TIFS.2018.2885282
8. NortonLifeLock: Uber Announces New Data Breach Affecting 57 million Riders and Drivers. https://us.norton.com/internetsecurity-emerging-threats-uber-breach-57-million.html (2020), retrieved: June 17, 2021
9. Pew Research Center: More Americans Are Using Ride-Hailing Apps. https://www.pewresearch.org/fact-tank/2019/01/04/more-americans-are-using-ride-hailing-apps/ (2019), retrieved: June 17, 2021
10. Pham, A., Dacosta, I., Endignoux, G., Troncoso-Pastoriza, J.R., Huguenin, K., Hubaux, J.: ORide: A Privacy-Preserving yet Accountable Ride-Hailing Service. In: Kirda, E., Ristenpart, T. (eds.) 26th USENIX Security Symposium, USENIX Security 2017, Vancouver, BC, Canada, August 16-18, 2017. pp. 1235-1252. USENIX Association (2017), https://www.usenix.org/conference/usenixsecurity17/technical-sessions/presentation/pham
11. Pham, A., Dacosta, I., Jacot-Guillarmod, B., Huguenin, K., Hajar, T., Tranèr, F., Gligor, V.D., Hubaux, J.: PrivateRide: A Privacy-Enhanced Ride-Hailing Service. PoPETs 2017(2), 38–56 (2017). https://doi.org/10.1515/popets-2017-0015
12. Schneider, T., Treiber, A.: A comment on privacy-preserving scalar product protocols as proposed in “spoc”. IEEE Transactions on Parallel and Distributed Systems 31(3), 543–546 (2020). https://doi.org/10.1109/TPDS.2019.2939313
13. The Sage Developers: SageMath, the Sage Mathematics Software System (Version 9.0) (2021). https://www.sagemath.org
14. thejournal.ie: West Dublin gang using hailing apps to target older taxi drivers. https://www.thejournal.ie/west-dublin-taxi-robbery-4420178-Jan2019/ (2019), retrieved: June 17, 2021
15. Vivek, S.: Attacks on a privacy-preserving publish-subscribe system and a ride-hailing service. CoRR abs/2105.04351 (2021), https://arxiv.org/abs/2105.04351, to appear in IMACC 2021.
16. Wang, F., Zhu, H., Liu, X., Lu, R., Li, F., Li, H., Zhang, S.: Efficient and privacy-preserving dynamic spatial query scheme for ride-hailing services. IEEE Transactions on Vehicular Technology 67(11), 11084–11097 (2018)
17. Xie, H., Guo, Y., Jia, X.: A privacy-preserving online ride-hailing system without involving a third trusted server. IEEE Transactions on Information Forensics and Security 16, 3068–3081 (2021). https://doi.org/10.1109/TIFS.2021.3065832
18. Yu, H., Jia, X., Zhang, H., Shu, J.: Efficient and privacy-preserving ride matching using exact road distance in online ride hailing services. IEEE Transactions on Services Computing pp. 1–1 (2020). https://doi.org/10.1109/TSC.2020.3022875
19. Yu, H., Shu, J., Jia, X., Zhang, H., Yu, X.: lpride: Lightweight and privacy-preserving ride matching over road networks in online ride hailing systems. IEEE Trans. Vehicular Technology 68(11), 10418–10428 (2019). https://doi.org/10.1109/TVT.2019.2941761
20. Zhao, Q., Zuo, C., Pellegrino, G., Lin, Z.: Geo-locating Drivers: A Study of Sensitive Data Leakage in Ride-Hailing Services. In: 26th Annual Network and Distributed System Security Symposium, NDSS 2019, San Diego, California, USA, February 24-27, 2019. The Internet Society (2019). https://www.ndss-symposium.org/ndss-paper/geo-locating-drivers-a-study-of-sensitive-data-leakage-in-ride-hailing-services/