A Privacy-Preserving Online Ride-Hailing System Without Involving a Third Trusted Server

Hongcheng Xie, *Graduate Student Member, IEEE*, Yu Guo, *Member, IEEE*, and Xiaohua Jia, *Fellow, IEEE*

***Abstract*— The increasing popularity of Online Ride-hailing (ORH) services has greatly facilitated our daily travel. It enables a rider to easily request the nearest driver through mobile devices in a short time. However, existing ORH systems require the collection of users' location information and thus raise critical privacy concerns. While several privacy-preserving solutions for ORH service have been proposed, most of existing schemes rely on an additional trusted party to compute the distance between a rider and a driver. Such a security assumption cannot fully address the privacy concerns for practical deployment. In this paper, we present a new ride-matching scheme for ORH systems, which allows privacy-preserving and effective distance calculation without involving a third-party server. Our proposed scheme enables ORH systems to securely compute the user distance while protecting the location privacy of both riders and drivers. Specifically, we resort to state-of-the-art distance calculation techniques based on Road Network Embedding (RNE), and show how to uniquely bridge cryptographic primitives like Property-preserving Hash (PPH) with RNE in depth to support privacy-preserving ride-matching services. Moreover, we also propose an optimized design to improve the matching efficiency. We formally analyze the security strengths and implement the system prototype. Evaluation results demonstrate that our design is secure and efficient for ORH systems.**

***Index Terms*— Location-based matching, online ride-hailing, privacy-preserving hash, ride-matching.**

I. Introduction

OVER the past decade, we have witnessed a rapid growth of Online Ride-hailing (ORH) services, such as Uber [1], Didi [2], GrabTaxi [3], and Lyft [4]. Unlike traditional taxi services, ORH system can help riders efficiently find the nearest taxi (i.e., driver) instead of waiting on the street. Along with this trend, there is a growing number of people that use ORH services on a daily basis.

Despite the great convenience, ORH systems raise privacy concerns because of its intrinsic service model. In a traditional ORH service model, both riders and drivers are required to submit their real-time locations to the server for ride-matching. Since the ORH server is an untrusted entity, it can use the sensitive information to track the users' driving paths and infer their daily activities. Thus, it is necessary to ensure the location privacy of riders and drivers in ORH services.

In the literature, a few works [5]–[8] studied privacy-preserving ORH services with the protection of both riders and drivers locations. PrivateRide [5] is the first system to enable encrypted ride-matching in ORH systems. The follow-up design [6] leverages Homomorphic Encryption [9] to enhance the security guarantees for user privacy. Unfortunately, these schemes utilize Euclidean distance for ride-matching, which is not suitable for practical ORH services in road networks. The reason is that vehicles are constrained to travel along the roads, and thus the Euclidean distance cannot be used for measuring the real distance between riders and vehicles. To address this issue, pRide [7] proposed to use the Road Network Embedding (RNE) technique [10] to estimate the road distance and perform ride-matching, but the scheme assumes the existence of two non-collusion servers to perform the matching procedure via Yao's Garble Circuit [11]. The expensive bandwidth overhead between two non-collusion servers results in matching performance being affected by network quality. To improve the ride-matching efficiency, the authors also proposed to leverage trusted hardware technology (i.e., Intel SGX [12]) for secure distance calculation [8]. However, the security guarantee of their design relies on the existence of trusted hardware at the server-side. Therefore, to achieve secure ORH services without involving a trusted third-party, it remains an unsolved issue.

In this work, we design and implement a privacy-preserving ride-matching scheme for secure ORH service, which allows the ORH server to accurately match riders and drivers over the road networks without learning any informations of location. Our proposed scheme does not need a trusted third-party in ride-matching so that the ORH server can directly compute the distance between the rider and drivers over the encrypted data. In this paper, we utilize the RNE techniques [10] to compute the shortest road distance in plaintext. To compute the distance over encrypted locations without a third-party, we propose a bit-block encryption and difference evaluation scheme based on Property-preserving Hash (PPH) [13]. The basic idea is to encrypt RNE vectors into bit-blocks via bilinear mapping, and embed the bit weight with random masks in each block cipher. Thus, ORH server can leverage bilinear mapping operations to

Manuscript received June 24, 2020; revised January 12, 2021; accepted February 25, 2021. Date of publication March 12, 2021; date of current version May 6, 2021. This work was supported by the Fundamental Research Funds for the Central Universities 310421108, the Research Grants Council of Hong Kong under Project No. CityU 11202419, and the National Natural Science Foundation of China Key Project under Grant No. 61732022. The associate editor coordinating the review of this manuscript and approving it for publication was Prof. Nele Mentens. *(Corresponding author: Yu Guo.)*

Hongcheng Xie and Xiaohua Jia are with the Department of Computer Science, City University of Hong Kong, Hong Kong, SAR, China (e-mail: hongcheng.xie@my.cityu.edu.hk; csjia@cityu.edu.hk).

Yu Guo is with the School of Artificial Intelligence, Beijing Normal University, Beijing 100875, China (e-mail: yuguo@bnu.edu.cn).

Digital Object Identifier 10.1109/TIFS.2021.3065832

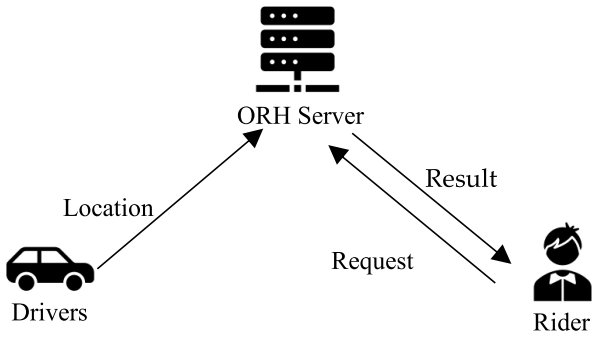


Fig. 1. System overview.

securely conduct distance comparisons and accurately match riders without involving a trusted third-party. In summary, our main contributions of this paper are as follows:

- We propose a secure Online Ride-hailing framework that the server can find the most suitable driver for each rider without involving a trusted third party.
- We devise a bit-block encryption method that can compute the difference of two encrypted values.
- We implement a system prototype and evaluate the performance using real-world datasets. Evaluation results show that our proposed design can achieve a better performance than the state-of-the-art privacy-preserving scheme [7].

The rest of this paper is organized as follows: Section II describes the system model and formalizes the target problem. Section III introduces some necessary preliminaries. We present the details of our proposed system in Section IV. We analyze the complexity and security in Section V and evaluate the performance in Section VI. Section VII reviews the related work. Finally, we conclude the whole paper in Section VIII.

II. Problem Statement

A. System Architecture

The system consists of three entities: the *ORH server*, a *rider* and multiple *drivers*, as shown in Fig. 1. In our system architecture, rider and drivers send their encrypted location data to the server, where ride-matching is then conducted in the encrypted domain at the server-side. The detailed roles are defined as follows:

- *ORH Server:* The ORH server is responsible for securely calculating the estimated distances and matching the nearest driver for the rider. It stores encrypted location sets from drivers and calculates the road distances when receiving the rider's request. At last, it selects the driver with the shortest distance as the most suitable one.
- *Rider And Drivers:* They are the people who use the ORH system. To enjoy secure ride-matching services, both rider and drivers need to convert their original locations into a set of road vectors by using RNE and encrypt the RNE vectors via our proposed cryptographic primitives. Then they upload the ciphertexts to the ORH server for the ride-matching processes. After secure distance calculation, the rider can receive the nearest driver's information from the server.

TABLE I
Notations

Notation	Definition
$RN = (V, E)$	the road network
$V_{i,j}$	a subset of V with 2^i nodes
R	$R = \{V_{1,1}, ..., V_{1,\alpha}, ..., V_{\beta,1}, ..., V_{\beta,\alpha}\}$
$dist(u, v)$	the shortest distance between u and v
$\delta(o_r, o_d)$	the estimated distance between o_r and o_d
m	the number of the blocks
l	the length of binary block
H, F	two hash functions $\{0,1\}^* \rightarrow Z_p$
$S(o)$	the location vector of object o
$S_i(o)$	the i-th component of $S(o)$
$\|S\|$	the number of components in vectors
$v_{\|i}$	the value of $(i+1)$-th block in one value
$\hat{S}(o)$	the ciphertext set $S(o)$
$\hat{v}$	the ciphertext of one value v
$\hat{v}_{\|i}$	the ciphertext of the $(i+1)$-th block of one value v
$\hat{v}_{\|i,z_j}$	the ciphertext of z_j of v's $(i+1)$-th block
P, Q	the PPH ciphertext
CT	the masked difference
T	the token for mask generation

B. Threat Model

In this system, we consider the ORH server is *honest-but-curious*. It strictly follows the pre-defined service protocols but it may intend to learn the private information about the riders' and drivers' locations. The attacker inside the ORH server can monitor the encrypted RNE vectors and the ride-matching result. We do not consider the case that a malicious attacker may modify or delete the vector ciphertexts. The rider and drivers are fully trusted that securely maintain the private keys for the ciphertext generation. The information of the road network is public so that everyone can access the road network data without permission.

C. Design Goals

Under the system architecture and threat model described above, the design goals of our research are listed as follows.

- *Security Guarantees*: The proposed scheme should ensure strong protection on the location privacy during the service flow. The ORH server should never learn the content of location data.
- *Matching Accuracy And Efficiency*: The scheme should provide accurate ride-matching results and be efficient to support ride-matching in a real-time fashion. The ORH server should find the nearest driver with a high probability for every rider by using the road distance. Meanwhile, its matching latency and communication cost should be bounded to enable large-scale ORH services.
- *No Need of Trusted Third-Party:* The ORH server should be able to conduct secure ride-matching without involving a trusted third-party.

III. Preliminaries

A. Road Network Embedding

Road Network Embedding (RNE) [10] provides a scheme to abstract the road network as a high-dimensional space.

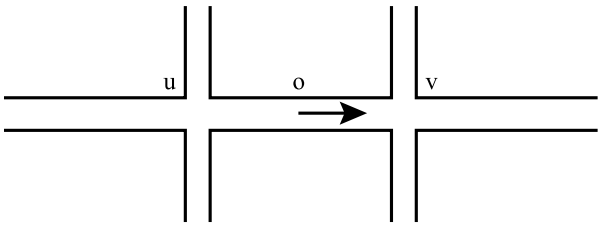


Fig. 2. The moving object o between u and v.

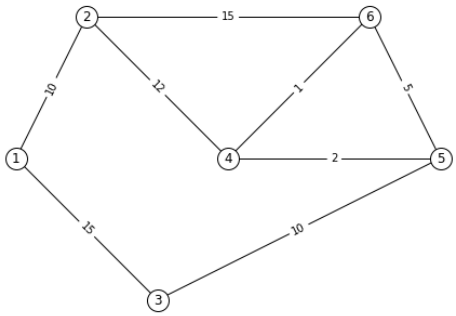


Fig. 3. An example of road network.

It allows the system to represent a location with a road vector and calculate the approximate distance between two locations. The road network is defined as $RN = (V, E)$, where V is the set of road nodes and E is the set of road segments. Let $|V|$ denotes the number of nodes in V.

Let n represents the number of the nodes in V, i.e., $n = |V|$. We define a set R consists of the subsets of V, i.e., $R = \{V_{1,1}, \ldots, V_{1,\alpha}, .., V_{\beta,1}, \ldots, V_{\beta,\alpha}\}$, where $\alpha = O(logn)$, $\beta = O(logn)$. Each subset $V_{i,j}$ consists of 2^i nodes chosen from V randomly. Let $dist(u, u')$ denote the shortest distance between node u and u' in rode network, and let $dist(u, V_{i,j})$ denote the minimum of $dist(u, u')$, where $u' \in V_{i,j}$. Hence, the location vector of the road node u could be defined as follows:

$$S(u) = (S_{V_{1,1}}(u), \ldots, S_{V_{1,\alpha}}(u), \ldots, S_{V_{\beta,1}}(u), \ldots, S_{V_{\beta,\alpha}}(u)) \tag{1}$$

where $S_{V_{i,j}}(u) = dist(u, V_{i,j})$.

Given an object o (i.e., a rider o_r or a driver o_d) which is between two road nodes $\{u, v\}$ as shown in Fig. 2, the location vector of o can be calculated as:

$$S_{V_{i,j}}(o) = min\{S_{V_{i,j}}(u) + dist(o, u), \times S_{V_{i,j}}(v) + dist(o, v)\} \tag{2}$$

Given two vectors of the locations of the rider o_r and driver o_d, the shortest distance $dist(o_r, o_d)$ between o_r and o_d can be approximated by the RNE estimated distance. Let $\delta(o_r, o_d)$ denote the estimated distance between o_r and o_d, where $dist(o_r, o_d) \approx \delta(o_r, o_d)$. $\delta(o_r, o_d)$ can be calculated by computing the maximum of the absolute values of the dimension differences, i.e.,

$$\delta(o_r, o_d) = max_{1 \le i \le \beta, 1 \le j \le \alpha}(|S_{V_{i,j}}(o_r) - S_{V_{i,j}}(o_d)|). \tag{3}$$

For instance, Fig. 3 consists of six road nodes. Eq. 4 is the shortest distance matrix $A = \{a_{i,j}\}$ of this road network, where $a_{i,j}$ is the shortest distance between node i and node j. As there are 6 road nodes, we set $\alpha = \beta = 2$. Thus, the set R should have four subsets of V, i.e., $R = \{V_{1,1}, V_{1,2}, V_{2,1}, V_{2,2}\}$, where $\{V_{1,1}, V_{1,2}\}$ have $2^1 = 2$ nodes chosen from V and $\{V_{2,1}, V_{2,2}\}$ have $2^2 = 4$ nodes chosen from V randomly. We assume that $V_{1,1} = \{1, 3\}$, $V_{1,2} = \{4, 5\}$, $V_{2,1} = \{1, 3, 4, 6\}$, and $V_{2,2} = \{2, 4, 5, 6\}$.

$$A = \begin{pmatrix} 0 & 10 & 15 & 22 & 24 & 23 \\ 10 & 0 & 24 & 12 & 14 & 13 \\ 15 & 24 & 0 & 12 & 10 & 13 \\ 22 & 12 & 12 & 0 & 2 & 1 \\ 24 & 14 & 10 & 2 & 0 & 3 \\ 23 & 13 & 13 & 1 & 3 & 0 \end{pmatrix} \tag{4}$$

Suppose a rider is on the road between node 1 and node 2. The distance between this rider o_r and node 1 is 4 and the distance between o_r and node 2 is 6. To calculate o_r's RNE vector, it should use the RNE vectors of node 1 and 2.

For node 1, $S_{V_{i,j}}(1)$ is the smallest shortest distance between node 1 and the nodes in $V_{i,j}$. The shortest distances between node 1 and the nodes in $V_{1,1}$ are 0 and 15. As the smallest value is 0, $S_{V_{1,1}}(1) = 0$. Similarly, $S_{V_{1,2}}(1) = 22$, $S_{V_{2,1}}(1) = 0$, and $S_{V_{2,2}}(1) = 10$. Finally, the RNE vector of node 1 is $(0, 22, 0, 10)$. Similarly, for node 2, $S_{V_{1,1}}(2) = 10$, $S_{V_{1,2}}(2) = 12$, $S_{V_{2,1}}(2) = 10$, and $S_{V_{2,2}}(2) = 0$. Finally, the RNE vector of node 2 is $(10, 12, 10, 0)$.

Since the distance between the rider o_r and node 1 and node 2 are 4 and 6 respectively, we can get that $S_{V_{1,1}}(o_r) = min\{4 + 0, 6 + 10\} = 4$ according to Eq. 2. Similarly, $S_{V_{1,2}}(o_r) = min\{4 + 22, 6 + 12\} = 18$, $S_{V_{2,1}}(o_r) = min\{4 + 0, 6 + 10\} = 4$, and $S_{V_{2,2}}(o_r) = min\{4 + 10, 6 + 0\} = 6$. Finally, the RNE vector of rider o_r is $(4, 18, 4, 6)$.

The RNE distance between two locations can be evaluated as shown in Eq. 3. For example, the RNE distance between node 1 and node 2 is $max\{|10 - 0|, |12 - 22|, |10 - 0|, |0 - 10|\} = 10$.

Note that by using RNE, we convert the computing of distance between two objects as the minus operation between the components of their location vectors and then finding the maximum of the results.

B. Bilinear Map

Bilinear Map is a map $e : G_1 \times G_2 \rightarrow G_T$, where G_1, G_2 and G_T are all multiplicative cyclic groups with the same prime order p. g_1 is the generator of G_1 and g_2 is the generator of G_2. The bilinear map e meets the following properties:

- there exists an efficiently computable algorithm for computing the map e.
- $e(g_1^a, g_2^b) = e(g_1, g_2)^{ab}$, for $\forall a, b \in Z$.
- $e(g_1, g_2) \neq 1$, for $g_1 \in G_1$ and $g_2 \in G_2$.

C. Property-Preserving Hash

Property-Preserving Hash (PPH) [13], [14] is a searchable encryption scheme. Server can use bilinear mapping operations to reveal the orders of the PPH ciphertexts. The property P is

defined as follows,

$$P(x, x^*) = \begin{cases} 1 & x = x^* + 1 \\ 0 & otherwise \end{cases}$$

Given a bit value x, the PPH ciphertext of x is defined as:

$$\mathrm{PPH(x)} = (\mathrm{g_1^{r_1}}, \mathrm{g_1^{r_1 \times H(k,x)}}, \mathrm{g_2^{r_2}}, \mathrm{g_2^{r_2 \times H(k,x+1)}}) \quad (5)$$

where $g_1 \in G_1$ and $g_2 \in G_2$. $\{r_1, r_2\} \in Z_p$ are random nonces chosen from Z_p and k is the private key.

PPH(x) can be denoted as $\mathrm{PPH(x)} = (\mathrm{P_1}, \mathrm{Q_1}, \mathrm{P_2}, \mathrm{Q_2})$, where $P_1 = g_1^{r_1}$, $Q_1 = g_1^{r_1 \times H(k,x)}$, $P_2 = g_2^{r_2}$ and $Q_2 = g_2^{r_2 \times H(k,x+1)}$. The tuple (P_1, Q_1) can be regarded as the ciphertext of x and (P_2, Q_2) can be regarded as the ciphertext of $x + 1$. PPH(x) includes the ciphertexts of x and $x + 1$.

The PPH ciphertext has an important property. Given two PPH ciphertexts $\mathrm{PPH(x)} = (\mathrm{P_1}, \mathrm{Q_1}, \mathrm{P_2}, \mathrm{Q_2})$ and $\mathrm{PPH(x^*)} = (\mathrm{P_1^*}, \mathrm{Q_1^*}, \mathrm{P_2^*}, \mathrm{Q_2^*})$, a server can test whether $P(x, x^*) = 1$ by checking

$$e(P_1, Q_2^*) = e(Q_1, P_2^*). \quad (6)$$

We call it matching if Eq. 6 holds, i.e., x matches $x^* + 1$. By using this property, we can determine the order of two 1-bit values easily.

IV. SECURE ORH SERVICE

A. Secure Value Comparison Based on PPH

To enable privacy-preserving ORH service, both rider o_r and drivers o_d need to encrypt their locations before submitting to the server, and the server has to compute the RNE estimated distance between o_r and o_d over the encrypted data. Without involving a trusted third-party, the most challenging issue is to compute the difference between two encrypted values, and find the maximum among the differences, as shown in Eq. 3. Existing homomorphic encryption schemes [9] allows computing the difference of two encrypted values but it cannot support encrypted order comparison and it incurs high computational cost. While some cryptographic primitives such as order-preserving encryption schemes [15]–[20] allow order comparison on ciphertexts, but they are not able to compute the difference between two ciphertexts. To this end, we propose a scheme based on the PPH scheme [13], [14]. PPH ciphertext inherits the order relations of the underlying values so that comparisons can be directly applied. However, the PPH scheme cannot be applied directly to securely compute the distance. As shown in Eq. 5, given a 1-bit value, its PPH ciphertext consists of the ciphertexts of two values. Therefore, given a long-bit value, expressing all the possible differences via the PPH scheme will cause a significant increase in the number of ciphertexts, rending huge storage overhead.

Our task is to devise a new scheme that allows the server to compute the difference between two values without knowing the plaintext of the values. Let v and v^* be two integers. We can regard v as a component of the location vector of the rider, and v^* as a component of the vector of the driver. Our main idea is to divide v and v^* into binary bit-blocks with equal length and encrypt each block with weighted differences by using PPH schemes. Then, we match the block-wise ciphertext of v^* with that of v, and obtain the block-wise differences of v^* and v in ciphertext. Finally, we compute the difference between values v^* and v. The details are discussed below.

Both v and v^* are divided into bit-blocks of fixed bit length l. Let $\{v_{|m-1}, v_{|m-2}, \ldots, v_{|0}\}$ and $\{v^*_{|m-1}, v^*_{|m-2}, \ldots, v^*_{|0}\}$ denote the bit-block sequences of v and v^* respectively, where m is the number of the bit-blocks, and $v_{|i}$(and $v^*_{|i}$) is the $(i+1)$-th block of v(and v^*) from the least significant bit. Let w_i denote the weight of the $(i+1)$-th block. Since the $(i+1)$-th block has l bits, its weight is $w_i = (2^l)^i$. For example, a binary value $v =$ "101″ can be divided into three blocks $v_{|2} = 1$, $v_{|1} = 0$ and $v_{|0} = 1$. As the length is 1 bit, their weights are $w_2 = 2^2$, $w_1 = 2^1$ and $w_0 = 2^0$.

We encrypt each bit-block of v as below. Let Z denote a set of all binary numbers from 0 to $2^l - 1$, i.e., $Z = \{z_j | z_j \in [0, 2^l - 1]\}$, where z_j is a binary number of bit length l. For bit-block $v_{|i}$, let the difference $d_{j,i} = (z_j - v_{|i}) \times w_i$, where $z_j \in Z$ and w_i is the weight of $v_{|i}$. For each z_j, it is uniquely associated with $d_{j,i}$. Let $v_{|i,z_j} = (z_j, d_{j,i})$. We represent $v_{|i}$ by a set of pairs $\{v_{|i,z_j} = (z_j, d_{j,i}) | z_j \in Z\}$, i.e., each z_j in the set Z and its weighted difference $d_{j,i}$. For example, given a bit-block $v_{|1} =$ "10″, $v_{|1}$ can be represented by {("00″, $(-2) * (2^2)^1$), ("01″, $(-1) * (2^2)^1$), ("10″, $0 * (2^2)^1$), ("11″, $1 * (2^2)^1$)}. Let $\hat{v}_{|i}$ denote the ciphertext of block $v_{|i}$ and $\hat{v}_{i,z_j}$ denote the ciphertext of pair $(z_j, d_{j,i})$. We encrypt $v_{|i}$ by encrypting $\{(z_j, d_{j,i})\}$, i.e.,

$$\hat{v}_{|i} = \{\hat{v}_{|i,z_j} = (\hat{z}_j, \hat{d}_{j,i}) | z_j \in Z\} \quad (7)$$

where $\hat{z}_j$ is the PPH ciphertext of z_j and $\hat{d}_{j,i}$ is the masked value of $d_{j,i}$. We will discuss about how to mask the difference $d_{j,i}$ later.

As shown in Eq. 7, the ciphertext of $v_{|i}$, i.e., $\hat{v}_{|i}$ consists of a set of ciphertexts $\hat{v}_{|i,z_j}$, one associated with each $z_j \in Z$. Note that the order of all $\hat{v}_{|i,z_j}$ in $\hat{v}_{|i}$ can be shuffled.

As for the encryption of $v^*_{|i}$, we can simply encrypt $v^*_{|i}$ by PPH. Since $v^*_{|i}$ is a bit-block of length l, $v^*_{|i}$ must be a value in set Z. Suppose $v^*_{|i} = z_{j^*}$, the PPH ciphertext of $v^*_{|i}$ must match $\hat{z}_{j^*}$ in Eq. 7 by using Eq. 6. Thus, the corresponding difference $d_{j^*,i}$ can be obtained, which is the difference between the $(i+1)$-th block values of v and v^*. According to the property of PPH scheme, only z_{j^*} in the set Z can match $v^*_{|i}$.

To protect the weighted difference $d_{j,i}$ in $v_{|i,z_j}$, $d_{j,i}$ should be masked to $\hat{d}_{j,i}$ with a hash value, and only the $\hat{d}_{j^*,i}$ which $v^*_{|i} = z_{j^*}$ can be unmasked. Let $\hat{v}^*_{|i}$ denote the ciphertext of $v^*_{|i}$ in our proposed ORH service. Each mask in $v_{|i,z_j}$ should be associated with z_j and order i. $\hat{v}^*_{|i}$ should include a token associated with $v^*_{|i}$ and order i besides the PPH ciphertext. We can use the token in $\hat{v}^*_{|i}$ to unmask $\hat{d}_{j^*,i}$ if $v^*_{|i} = z_{j^*}$ holds. The server can unmask the mask of $d_{j^*,i}$ and obtain the value of $d_{j^*,i}$ without knowing the plaintext of $v_{|i}$ and $v^*_{|i}$.

For each pair $(z_j, d_{j,i})$ of block $v_{|i}$, its ciphertext $\hat{v}_{|i,z_j}$ is defined as

$$(g_1^{r_1}, g_1^{r_1 \times H(k_1, z_j)}, F(H(k_2, z_j || i), \gamma_j) \oplus d_{j,i}, \gamma_j), \quad (8)$$

where r_1 and γ_j are two fresh nonces, H and F are two hash functions, and k_1 and k_2 are private keys shared among the

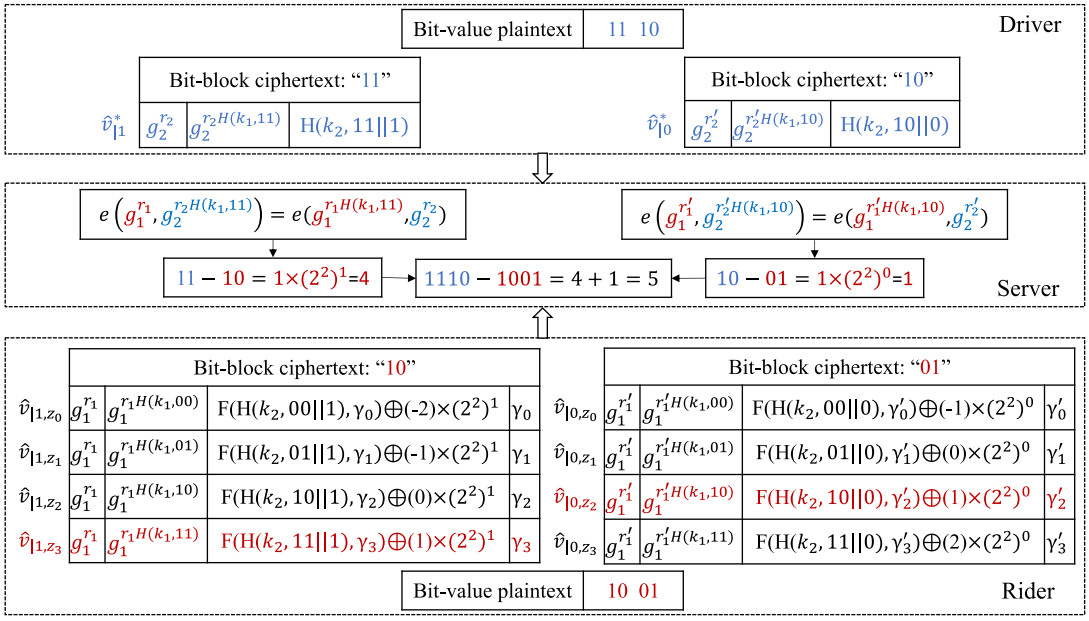


Fig. 4. An example of computing difference between two integers over ciphertext.

users, i.e., riders and drivers. In our design, they are distributed by a Key Authority for the secure ORH service. Key Authority is an admittance control entity, which does not participate in ORH services. As demonstrated in Section II-B, we consider that riders and drivers are always trusted. They will not expose private keys to other parties. Meanwhile, the detection of this security concern can be addressed via other complementary techniques like encryption key rotation [21], [22].

According to Eq. 8, $\hat{z}_j = (g_1^{r_1}, g_1^{r_1 \times H(k_1,z_j)})$ and $\hat{d}_{j,i} = (H(k_2, z_j||i), \gamma_j) \oplus d_{j,i}, \gamma_j)$, which $F(H(k_2, z_j||i), \gamma_j)$ is the mask. $\hat{v}_{|i}^*$ includes the token $H(k_2, v_{|i}^*||i)$ so that the mask $F(H(k_2, z_{j^*}||i), \gamma_{j^*})$ can be calculated by using $H(k_2, v_{|i}^*||i)$ from $\hat{v}_{|i}^*$ and γ_{j^*} from $\hat{v}_{i,z_{j^*}}$ if $v_{|i}^* = z_{j^*}$.

As for value v^*, which is a component in driver's location vector, we encrypt each of its block $v_{|i}^*$ simply. Its ciphertext $\hat{v}_{|i}^*$ is defined as

$$(g_2^{r_2}, g_2^{r_2 \times H(k_1, v_{|i}^*)}, H(k_2, v_{|i}^*||i)) \tag{9}$$

where r_2 is the fresh nonce, $H(k_2, v_{|i}^*||i)$ is the token, and $(g_2^{r_2}, g_2^{r_2 \times H(k_1, v_{|i}^*)})$ is the PPH ciphertext of $v_{|i}^*$. The ciphertext of value v^* consists of the ciphertexts of all the blocks $v_{|i}^*$.

Fig. 4 describes an example of computing difference between two integers over ciphertext. In rider's side, a value 9 ("1001″) is divided into two blocks "10″ and "01″. For the block $v_{|1} =$ "10″ $= 2$, four values z_j from 0 to $(2^2 - 1 = 3)$ are encrypted according to Eq. 8. For $z_0 = 0$, as the length of each block is 2, its weight is $(2^2)^1$. Its PPH ciphertext and the difference $(d_{0,1} = (0 - 2) \times (2^2)^1 = -2 \times (2^2)^1)$ between z_0 and $v_{|1}$ are calculated. The difference is masked by XORing the mask $F(H(k_2, 00||1), \gamma_0)$. Similarly, the other z_j are encrypted to build the bit-block ciphertext of "10″. The encryption of block "01″ is processed by the same procedure. At last, the ciphertexts of "10″ and "01″ compose the ciphertext of 9 ("1001″).

In driver's side, value 14 ("1110″) is divided into two blocks "11″ and "10″. For the block $v_{|1}^* =$ "11″ $= 3$, only the PPH ciphertext of "11″ and the hash token $H(k_2, 11||1)$ which is used for mask generation in difference calculation are calculated. The bit-block ciphertext of "10″ is generated similarly. At last, the ciphertexts of "11″ and "10″ compose the ciphertext of "1110″.

Given two ciphertexts $\hat{v}_{|i}$ and $\hat{v}_{|i}^*$, the server finds z_{j^*} which $z_{j^*} = v_{|i}^*$ by testing whether the following equation holds with all $\hat{v}_{|i,z_j} \in \hat{v}_{|i}$ and $\hat{v}_{|i}^*$:

$$e(g_1^{r_1}, g_2^{r_2 \times H(k_1, v_{|i}^*)}) = e(g_1^{r_1 \times H(k_1, z_j)}, g_2^{r_2}), \tag{10}$$

where $g_1^{r_1}$ and $g_1^{r_1 \times H(k_1, z_j)}$ are from $\hat{v}_{|i,z_j}$, and $g_2^{r_2}$ and $g_2^{r_2 \times H(k_1, v_{|i}^*)}$ are from $\hat{v}_{|i}^*$.

The difference between z_{j^*} and $v_{|i}$ in z_{j^*}'s ciphertext will be the difference between $v_{|i}^*$ and $v_{|i}$ if z_{j^*} matches $v_{|i}^*$. According to the above discussion, the hash token in driver's ciphertext $H(k_2, v_{|i}^*||i)$ is equal to $H(k_2, z_{j^*}||i)$ in the mask of z_{j^*}. Therefore, the weighted difference of the $(i + 1)$-th pair of blocks can be revealed by XORing the random masks $F(H(k_2, v_{|i}^*||i), \gamma_{j^*})$, where $H(k_2, v_{|i}^*||i)$ is from driver's token and γ_{j^*} is from rider's ciphertext. Finally, the difference between two values v^* and v can be calculated by summing all the weighted differences as:

$$v^* - v = \sum_{i=0}^{n-1} d_{j^*,i}, \tag{11}$$

where $d_{j^*,i}$ is the weighted difference of z_{j^*} which z_{j^*} is the matching value in the $(i + 1)$-th block.

Fig. 4 illustrates the difference calculation procedure in server. For bit-block "11″ in "1110″, the server test whether

Eq. 10 holds or not with the PPH ciphertext of "11″ and each PPH ciphertext of z_j in the ciphertext set of "10″. We can easily find that the PPH ciphertext of "11″ matches that of z_3 = "11″ by checking $e(g_1^{r_1}, g_2^{r_2 H(k_1,11)}) = e(g_1^{r_1 H(k_1,11)}, g_2^{r_2})$, where $g_1^{r_1}$ and $g_1^{r_1 H(k_1,11)}$ are from $\hat{v}_{|1,z_3}$ in bit-block ciphertext "10″ from rider, and $g_2^{r_2}$ and $g_2^{r_2 H(k_1,11)}$ are from $\hat{v}_{|1}$ in bit-block ciphertext "11″ from driver, as highlighted in Fig. 4. The server calculates the mask $F(H(k_2, 11||1), \gamma_3)$ by using the hash value with $H(k_2, 11||1)$ in bit-block token of "11″, and γ_3 in z_3's ciphertext. After XORing the masked weighted difference with the mask value, the weighted difference $1 \times (2^2)^1$ between "11″ and "10″ can be revealed. The weighted difference 1 between the second block pairs "10″ and "01″ can also be revealed in the same way. At last, the server can calculate the difference 5 between 14 ("1110″ in binary) and 9 ("1001″ in binary) by summing the weighted differences together according to Eq. 11.

Based on our tailored ride-matching scheme, the ORH server can calculate the difference between two components while protecting the location privacy. Thus, it can compute the road distance between a rider and each driver from the location vectors and select the closest driver for each rider.

B. Secure Ride-Matching Over Encrypted Locations

In ORH service, the ORH server will find the nearest driver for each rider based on their uploaded locations. According to Sec. III-A, the location will be embedded in a location vector. For convenience, we rewrite the location of u in Eq. 1 as:

$$S(u) = (S_1(u), S_2(u), \ldots, S_\eta(u)), \tag{12}$$

where $\eta = \alpha \times \beta$.

Suppose there are multiple drivers o_{d_i} and a rider o_r. To find the nearest driver, the ORH server will evaluate the distance $\delta(o_r, o_{d_i})$ between o_r and each driver o_{d_i}, and return the driver with the smallest distance.

According to Eq. 3, we find that the distance between o_r and o_{d_i} is the maximum of the absolute values of the dimension differences. Hence, we can consider each component in the location vector as an individual value. Each component $S_k(o_r)$ in o_r's location vector $S(o_r)$ is encrypted into $\hat{S}_k(o_r)$ in the same way as the value v as discussed in Sec. IV-A. Each component $S_k(o_{d_i})$ in o_{d_i}'s location vector o_{d_i} is encrypted into $\hat{S}_k(o_{d_i})$ in the same way as the value v^* as discussed in Sec. IV-A. The rider and drivers submit the ciphertext of their location vectors to the server. The server can calculate the difference $(S_k(o_r) - S_k(o_{d_i}))$ by using Eq. 11. As the result is the plaintext value of $S_k(o_r) - S_k(o_{d_i})$, it can be processed by the arithmetic operations in Sec. III-A directly. According to Eq. 3, the server can calculate the absolute value of each $(S_k(o_r) - S_k(o_{d_i}))$ and get the maximum value as $\delta(o_r, o_{d_i})$.

The server calculates the distance between o_r and each driver o_{d_i} based on the above procedure. As the distance between the rider and each driver has been calculated, the driver with the smallest distance o_{d^*} is assigned to the rider by the ORH server. Since the location in road network is embedded, the estimated distance cannot leak the location information. The server can find the closest driver without knowing their locations.

Algorithm 1: BlockEncRider: Encrypt Rider's Block Value $v_{|i}$

Input: Private keys $\{k_1, k_2\}$, secure PRF $\{F, H\}$, the block value from a rider $v_{|i}$, $i \in [0, m-1]$.
Output: The block ciphertexts $\hat{v}_{|i}$.
Initialize a set $\hat{v}_{|i}$;
for $z_j \in [0, 2^l - 1]$ **do**
  Generate the ciphertext $(P_j, Q_j, CT_j, \gamma_j)$ of z_j according to Eq. 8
  $\hat{v}_{|i}$.append$(P_j, Q_j, CT_j, \gamma_j)$;
end
shuffle the order in $\hat{v}_{|i}$;
return $\hat{v}_{|i}$

C. Construction of Secure Ride-Matching Scheme

1) System Initialization: In the initialization step, the server first needs to construct the embedded road network and presets the system security parameters. Specifically, it broadcasts the location vector R and the set V based on the road network $RN = (V, E)$. Then, the server generates the cyclic groups G_1, G_2, and G_T, where g_1 is the generator of G_1 and g_2 is the generator of G_2. Meanwhile, the private keys k_1 and k_2 are shared among the riders and drivers. They are distributed by a Key Authority in ORH service, as described in Sec. IV-A. As the ORH server cannot learn k_1 and k_2, it cannot generate the dummy ciphertext, i.e., the ciphertext of server-known location, to infer the drivers' and riders' locations.

2) Secure Difference Computing: Both riders and drivers run the following algorithms to encrypt their location data v (or v^*), and the server can accordingly calculate the difference $v^* - v$.

BlockEncRider($v_{|i}$): Given a block value $v_{|i}$, a rider needs to run this algorithm to encrypt it locally. As shown in Alg. 1, $v_{|i}$ is represented by a set of pairs $\{v_{|i,z_j} = (z_j, d_{j,i}) | z_j \in Z\}$. Let CT_j denote the masked weighted difference of z_j. The ciphertexts $(P_j, Q_j, CT_j, \gamma_j)$ of all $v_{|i,z_j}$ are generated according to Eq. 8. Then, the algorithm returns the set $\hat{v}_{|i} = \{(P_j, Q_j, CT_j, \gamma_j)\}$ as the ciphertexts of this block.

ValueEncRider(v): Given a location value v of a rider, the algorithm first needs to pad 0 on the left to make sure that all values have the same length. Then the value v is divided into m blocks $\{v_{|m-1}, .., v_{|0}\}$ of l bits. Each block $v_{|i}$ is encrypted by using BlockEncRider($v_{|i}$), as shown in Alg. 2.

BlockEncDriver($v^*_{|i}$): Given a block value $v^*_{|i}$, a driver needs to run this algorithm to encrypt it locally. Let T_i denote the token of $v^*_{|i}$ for mask generation. The ciphertext (P_i^*, Q_i^*, T_i) of $v^*_{|i}$ is generated according to Eq. 9.

ValueEncDriver(v^*): Given a location value v^* of a driver, v^* is processed in the same way as v in Alg. 2. v^* is padded 0 on the left and is divided into m blocks $\{v^*_{|m-1}, .., v^*_{|0}\}$ of l bits. Each block $v^*_{|i}$ will be encrypted by using BlockEncDriver($v^*_{|i}$), as shown in Alg. 4.

Algorithm 2: ValueEncRider: Encrypt Rider's Value v

Input: The value from a rider v, bit length l.
Output: The encrypted value set $\hat{v}$.
Initialize a set $\hat{v}$;
if the length of v is smaller than the length limit **then**
  Padding 0 on the left until it is equal to the limit;
end
Divide v into m blocks $\{v_{|m-1}, .., v_{|0}\}$ of l bits
for each block $v_{|i}$, $i \in [0, m-1]$ **do**
  $\hat{v}$.append(BlockEncRider($v_{|i}$));
end
return $\hat{v}$

Algorithm 3: BlockEncDriver: Encrypt Driver's Block Value $v^*_{|i}$

Input: Private keys $\{k_1, k_2\}$, secure PRF $\{F, H\}$, the block value from a driver $v^*_{|i}$, $i \in [0, m-1]$.
Output: The block ciphertext $\hat{v}^*_{|i}$.
Generate the ciphertext $\hat{v}^*_{|i} = (P^*_i, Q^*_i, T_i)$ of $v^*_{|i}$ according to Eq. 9
return $\hat{v}^*_{|i}$

Algorithm 4: ValueEncDriver: Encrypt Driver's Value v^*

Input: The value from a driver v^*, bit length l.
Output: The encrypted value set $\hat{v}^*$.
Initialize a set $\hat{v}^*$;
if the length of v^* is smaller than the length limit **then**
  Padding 0 on the left until it is equal to the limit;
end
Divide v^* into m blocks $\{v^*_{|m-1}, .., v^*_{|0}\}$ of l bits
for each block $v^*_{|i}$, $i \in [0, m-1]$ **do**
  $\hat{v}^*$.append(BlockEncRider($v^*_{|i}$));
end
return $\hat{v}^*$

Algorithm 5: BlockCmp: Calculate Block Difference

Input: The block ciphertext $\hat{v}_{|i}$ from a rider and the block ciphertext $\hat{v}^*_{|i}$ from a driver.
Output: The weighted difference $d_{j,i}$.
$(P^*_i, Q^*_i, T_i) \leftarrow \hat{v}^*_{|i}$;
for each $\{P_j, Q_j, CT_j, \gamma_j\} \in \hat{v}_{|i}$ **do**
  if $e(P_j, Q^*_i) = e(Q_j, P^*_i)$ **then**
    Calculate the mask $F(T_i, \gamma_j)$
    Extract the weighted difference $d_{j,i}$ by $F(T_i, \gamma_j) \oplus CT_j$; **return** $d_{j,i}$;
  end
end

Algorithm 6: ValueCmp: Calculate Value Difference

Input: The query token $\hat{v}$ from a driver and the location ciphertexts $\hat{v}^*$ from a rider.
Output: The difference d.
for each ciphertext block pair $\{\hat{v}_{|i}, \hat{v}^*_{|i}\}$, $i \in [0, m-1]$ **do**
  $d = d$ + BlockCmp($\hat{v}_{|i}$, $\hat{v}^*_{|i}$);
end
return d

Algorithm 7: DistanceEst: Calculate Distance

Input: The encrypted vector set $\hat{S}(o_d)$ from a driver o_d and $\hat{S}(o_r)$ from a rider o_r.
Output: The estimated distance $\delta(o_d, o_r)$.
Initialize a set dSet;
for each $\hat{S}_k(o_r) \in \hat{S}(o_r)$ and corresponding ciphertext $\hat{S}_k(o_d) \in \hat{S}(o_d)$ **do**
  $d \leftarrow$ |ValueCmp($\hat{S}_k(o_r)$, $\hat{S}_k(o_d)$)|;
  Add d to the value set dSet
end
$\delta(o_r, o_d) \leftarrow$ the maximum of dSet;
return $\delta(o_r, o_d)$;

BlockCmp($\hat{v}_i$, $\hat{v}^*_i$): Given a block ciphertext $\hat{v}_{|i}$ from a rider and a block ciphertext $\hat{v}^*_{|i}$ from a driver, Alg. 5 can calculate the block difference. Specifically, it first uses the bilinear map operation to find the matched block, i.e., $e(P_j, Q^*_i) = e(Q_j, P^*_i)$. Then, the block difference can be revealed via computing $F(T_i, \gamma_j) \oplus CT_j$, where T_i is the block token $H(k_2, v^*_{|i}||i)$ in $\hat{v}^*_{|i}$ and γ_j is the random nonce in $\hat{v}_{|i,z_j}$. Thus, the weighted difference for this block can be calculated.

ValueCmp($\hat{v}$, $\hat{v}^*$): In this procedure, Alg. 6 can calculate the difference $v^* - v$ with the ciphertexts $\{\hat{v}, \hat{v}^*\}$. For each pair of block ciphertexts $\{\hat{v}_{|i}, \hat{v}^*_{|i}\}$, it first computes the weighted difference in each block via BlockCmp($\hat{v}_{|i}$, $\hat{v}^*_{|i}$) and then adds the weighted differences together according to Eq. 11. The final result is the difference between v^* and v.

3) RNE With Secure Difference Computing: In a RNE-based road network, both riders o_r and drivers o_d need to generate the location vector according to Eq. 1 and Eq. 2. Then, each component of the location vector is encrypted by using Alg. 6 and uploaded to the ORH server for secure distance calculation. Finally, the closest driver is recommended to the rider. Alg. 7 shows the detailed processes for distance calculation. Specifically, given encrypted vector sets $\hat{S}(o_d)$ and $\hat{S}(o_r)$, the ORH server first leverages the Alg. 6 to compute the difference set dSet, containing the difference d for each dimension pair $\{\hat{S}_k(o_r), \hat{S}_k(o_d)\}$. According to Eq. 3, the estimated distance $\delta(o_r, o_d)$ between o_r and o_d is the maximum value of the dSet set. After computing all the estimated distances, the ORH server selects the driver with the smallest distance as the most suitable driver.

D. Optimization of Matching Performance

According to RNE, the ORH server needs to calculate the difference between each pair of components in the rider o_r's and the driver o_{d_i}'s vector. Then it selects the maximum as the distance between o_r and the corresponding driver o_{d_i}. At last it selects the driver with the smallest distance as the matched driver. The ORH server has to complete the calculation task of each component for each driver's location. However, it is

sometimes obviously that the driver currently being processed has no hope of being selected. In the matching procedure, there is no need to wait for all the differences to select the maximum value as the distance. The distance calculation between rider and one driver sometimes can be terminated early if the server finds that the driver currently being processed is impossible to be chosen.

The ORH server calculates the distance between rider o_r and driver o_{d_i} one by one. We can use a variable to store the current minimum distance in the set of processed drivers. It could be replaced by the new result if the newly computed distance between the rider and the current processing driver is smaller than this variable.

Lemma 1: Suppose that cur is the distance between the rider and the closest driver in the set of processed drivers. During distance estimation between the rider o_r and a driver o_{d_i}, if any absolute value of difference $|S_k(o_{d_i}) - S_k(o_r)|$ between the pair of components $S_k(o_r) \in S(o_r)$ and $S_k(o_{d_i}) \in S(o_{d_i})$ is larger than cur, the distance $dist(o_r, o_{d_i})$ between rider o_r and driver o_{d_i} is larger than cur.

Proof: [Proof of lemma 1] As shown in Alg. 7, the distance between o_{d_i} and o_r is the maximum in the difference set. Therefore, the maximum of chessboard distance $\delta(o_r, o_{d_i})$ between $S(o_r)$ and $S(o_{d_i})$ must be larger than or equal to the difference calculated currently, i.e., $\delta(o_r, o_{d_i}) \geq |S_k(o_{d_i}) - S_k(o_r)|$. As $|S_k(o_{d_i}) - S_k(o_r)| > cur$, it could be deduced that $\delta(o_r, o_{d_i}) > cur$. ■

According to Lemma 1, the variable cur will not be replaced by distance $\delta(o_r, o_{d_i})$ if $|S_k(o_{d_i}) - S_k(o_r)| > cur$. Therefore, the distance calculation between rider o_r and driver o_{d_i} could be terminated if the conditions are met, as shown in Alg. 8. It checks whether the difference between two components is larger than the current smallest distance or not after the difference has been calculated. The distance calculation between o_r and o_{d_i} will be terminated early if it is larger than the current minimum value and this driver's result will not be considered in cur update. Thus, this driver will not be the most suitable driver for this rider. Section VI will show that this optimization could reduce the computation overhead effectively.

Algorithm 8: OptMatching: Optimized Matching

Input: The encrypted location vector of rider $\hat{S}(o_r)$, the set of encrypted location vectors of drivers $\{\hat{S}(o_{d_i})\}$.
Output: The matched driver: o_{d^*}.

```
cur ← MAX;
d* ← −1;
for each driver o_{d_i}'s encrypted vector Ŝ(o_{d_i}) do
    distance ← −1;
    for each k-th dimension's ciphertext Ŝ_k(o_r) in Ŝ(o_r) do
        get the ciphertext Ŝ_k(o_{d_i}) in Ŝ(o_{d_i});
        |S_k(o_{d_i}) − S_k(o_r)| ←
        DistanceEst(Ŝ_k(o_r),Ŝ_k(o_{d_i}));
        if |S_k(o_{d_i}) − S_k(o_r)| > cur then
            break
        end
        if |S_k(o_{d_i}) − S_k(o_r)| > distance then
            Update distance to |S_k(o_{d_i}) − S_k(o_r)|
        end
    end
    if the loop stops due to |S_k(o_{d_i}) − S_k(o_r)| > cur then
        do nothing
    end
    else
        if distance < cur then
            Update cur to distance Update d* to i;
        end
    end
end
return o_{d*};
```

TABLE II
COMPUTATION COMPLEXITY

Entity	Computation Complexity
Driver	$(2t_e + 2t_h + t_m) \cdot m \cdot \lvert S\rvert$
Rider	$(2t_e + 3t_h + t_m) \cdot 2^l \cdot m \cdot \lvert S\rvert$
ORH Server	$(((2^{l-1} \cdot t_b + t_h) \cdot m + m) \cdot \lvert S\rvert + \lvert S\rvert) \cdot n_d + n_d$

TABLE III
COMMUNICATION COMPLEXITY

Entity	Communication Complexity
Driver	$(2s_b + s_h) \cdot m \cdot \lvert S\rvert$
Rider	$(2s_b + s_h + s_r) \cdot 2^l \cdot m \cdot \lvert S\rvert$
ORH Server	0

V. COMPLEXITY AND SECURITY ANALYSIS

A. Complexity Analysis

In Table II and Table III, we analyze the computation complexity and communication complexity of our proposed scheme. In driver's encryption procedure, to generate the ciphertext (P, Q, T) of one bit-block, it needs to go through two exponentiation operations t_e, two hash operation t_h and one bignum mulitplication t_m. The ciphertext (P, Q, T) of one bit-block includes two bignum values s_b and one hash value s_h. Each component contains m encrypted bit-blocks and a driver's vector includes $|S|$ components.

During the rider's encryption procedure, 2^l ciphertexts will be generated for one bit-block rather than one ciphertext in driver's procedure. To generate one ciphertext in one bit-block, it needs to go through two exponentiation operations t_e, three hash operations t_h and one bignum mulitplication t_m. Each ciphertext (P, Q, CT, γ) includes two bignum values s_b, one hash value s_h and a random value s_r. The rider's bit-block ciphertext set consists of 2^l ciphertexts. Similarly, each component consists of m blocks and a rider's vector includes $|S|$ components.

For ORH server, it extracts the weighted difference between two corresponding blocks first. Given one driver's block, there are 2^l rider's ciphertexts to be matched. The average number of matches is 2^{l-1}. Each match needs to go through one

bilinear map operation t_b. If the match succeeds, the ORH server will perform a hash operation t_h to calculate the mask of difference. A component includes m bit-blocks. In addition, the ORH server needs to perform m basic arithmetic operations to add all the weighted differences together. The sum is the difference between two corresponding components. As the same as rider's and driver's encryption procedures, a vector includes $|S|$ values. Therefore, the ORH server performs the above procedure $|S|$ times to calculate the difference between each pair of values. The ORH server also needs to perform $|S|$ additional comparison operations to find the largest value of these differences as the distance between the rider and one driver. There are n_d drivers to be processed by the aforementioned scheme. At last, the ORH server needs to perform n_d comparison operations to select the driver with smallest distance as the most suitable driver. As there is no assistant server for comparison, the ORH server does not need to communicate with other server. Therefore, the communication cost of ORH server is 0.

B. Security Analysis

In this subsection, we present a rigorous security analysis to demonstrate that our ride-matching scheme achieves strong protection on location privacy. Firstly, we define the encryption leakage $\mathcal{L}^{\text{enc}}$ for a given value v as:

$$\mathcal{L}^{\text{enc}}(v) = (n, \langle |P^*|, |Q^*|, |CT|, \gamma \rangle),$$

where n is the number of blocks, $|P^*|, |Q^*|, |CT|$ are bit lengths of ciphertexts and γ is a random nonce. When the ORH server conducts ride-matching services, the view of an adversary is defined in the leakage $\mathcal{L}^{\text{cmp}}$ as:

$$\mathcal{L}^{\text{cmp}}(E, E^*) = (\langle P, Q, T \rangle, CT, N_{s\times s}),$$

where $\langle P, Q, T \rangle$ are tokens and CT is the matched block. $N_{s\times s}$ is a binary symmetric binary matrix that records repeated queries. Following the simulation-based security definition in [23], we give the formal security definition:

Definition 1: Let $\Omega = (\mathsf{BlockEncRider}, \mathsf{BlockEncDriver}, \mathsf{BlockCmp})$ be the secure ride-matching scheme, λ be the security parameter, and let $\mathcal{L}^{\text{enc}}$ and $\mathcal{L}^{\text{cmp}}$ be the leakage functions. We define the following probabilistic experiments $\mathbf{Real}_{\Omega,\mathcal{A}}(1^\lambda)$ and $\mathbf{Ideal}_{\Omega,\mathcal{A},\mathcal{S}}(1^\lambda)$ with a probabilistic polynomial time (PPT) adversary $\mathcal{A}$ and a PPT simulator $\mathcal{S}$:

$\mathbf{Real}_{\Omega,\mathcal{A}}(1^\lambda)$: $\mathcal{A}$ selects a value v and asks the user to generate the ciphertexts through the $\mathsf{BlockEncRider}$ and $\mathsf{BlockEncDriver}$ algorithms. Then $\mathcal{A}$ launches a polynomial number of queries and asks the user for the tokens and matched ciphertexts via $\mathsf{BlockCmp}$. Finally, $\mathcal{A}$ outputs a bit as the output.

$\mathbf{Ideal}_{\Omega,\mathcal{A},\mathcal{S}}(1^\lambda)$: $\mathcal{A}$ selects a value v, and $\mathcal{S}$ simulates a ciphertext for $\mathcal{A}$ based on $\mathcal{L}^{\text{enc}}$. Then $\mathcal{A}$ adaptively performs a polynomial number of queries. From the leakage $\mathcal{L}^{\text{cmp}}$ in each query, $\mathcal{S}$ simulates tokens and ciphertexts. Finally, $\mathcal{A}$ outputs a bit as the output.

Let $\mathcal{L} = (\mathcal{L}^{\text{enc}}, \mathcal{L}^{\text{cmp}})$, Ω is adaptively $\mathcal{L}$-secure for all PPT adversaries $\mathcal{A}$, there exists a simulator $\mathcal{S}$ such that: $Pr[\mathbf{Real}_{\Omega,\mathcal{A}}(1^\lambda) = 1] - Pr[\mathbf{Ideal}_{\Omega,\mathcal{A},\mathcal{S}}(1^\lambda) = 1] \leq negl(\lambda)$, where $negl(\lambda)$ is a negligible function in λ.

Theorem 1: Ω is an adaptive $\mathcal{L}$-secure scheme under the random-oracle model if F, H are secure PRFs.

Proof: [Proof of theorem 1] Our proposed design is essentially based on the PPH scheme in [24], except that we embed the weights into each block with random masks. We prove that the following games $\mathbf{Real}_{\Omega,\mathcal{A}}(1^\lambda)$ and $\mathbf{Ideal}_{\Omega,\mathcal{A},\mathcal{S}}(1^\lambda)$ are computationally indistinguishable for all polynomial-time adversaries $\mathcal{A}$. First, we describe a sequence of games that will be used in our security proof.

Game$_0$: Game_0 is exactly the real world searchable encryption security game $\mathbf{Real}_{\Omega,\mathcal{A}}$. It is clear that

$$Pr[\mathbf{Real}_{\Omega,\mathcal{A}}(1^\lambda) = 1] = Pr[\text{Game}_0 = 1] \tag{13}$$

Game$_1$: Game_1 is the same as Game_0 except that we pick random strings as the matched block CT for the value v rather that calling F. Game_1 records the mapping of the value v and the position of the matched block. When conducting the matching protocol, the position of the value v is required. It first retrieves the history to check if it has the corresponding block CT. If yes, it returns the block CT to the user. Otherwise, it randomly picks a string and denotes it as the matched block. It is also clear that, if there exists an adversary can distinguish the Game_1 and Game_0, then we can find an efficient solution S_1 to distinguish F and a truly random function. Namely,

$$Pr[\text{Game}_0 = 1] - Pr[\text{Game}_1 = 1] \leq \mathbf{Adv}^{prf}_{G,S_1}(\lambda). \tag{14}$$

Game$_2$: Game_2 is the same as Game_1 except that we pick random strings as the token T instead of generating it through secure PRF H. If there exists an adversary can distinguish Game_1 and Game_2, we can build an efficient solution S_2 to distinguish the key-based PRF H and a truly random function. Namely,

$$Pr[\text{Game}_1 = 1] - Pr[\text{Game}_2 = 1] \leq \mathbf{Adv}^{prf}_{G,S_2}(\lambda). \tag{15}$$

Game$_3$: Game_3 is exactly identical with Game_2 except that it guesses the encrypted item T. Intuitively, it is difficult to distinguish Game_2 and Game_3. Otherwise, an efficient solution S_3 can be found to break the security of the symmetric encryption scheme. It implies that

$$Pr[\text{Game}_2 = 1] - Pr[\text{Game}_3 = 1] \leq \mathbf{Adv}^{prf}_{Enc,S_3}(\lambda). \tag{16}$$

Essentially, the Game_3 is identical to $\mathbf{Ideal}_{\Omega,\mathcal{A},\mathcal{S}}$. Combining the above all games, it is not difficult to find that

$$|Pr[\mathbf{Real}_{\Omega,\mathcal{A}}] - Pr[\mathbf{Ideal}_{\Omega,\mathcal{A},\mathcal{S}}]| \leq \mathbf{Adv}^{prf}_{G,S_1}(\lambda) + \mathbf{Adv}^{prf}_{G,S_2}(\lambda) + \mathbf{Adv}^{prf}_{Enc,S_3}(\lambda). \tag{17}$$

This completes the proof. ■

VI. Experimental Evaluation

In this section, we evaluate the performance of our proposed system with the real-world data. We implement our non-optimized proposed system and the optimized construction with Road Network Embedding (RNE) [10]. We also use the experiment data of SHE-based pRide [7] as a contrast.

The secure distance computation module of our proposed system based on PPH was implemented by Pairing-Based Cryptography (PBC) Library proposed by Ben Lynn [25]. PBC Library is a pairing-based cryptography library that provides parameter management, group definition, an element I/O, element mathematical operations and bilinear map. There are different types of pairing parameters in PBC Library. In our experiment, we use Type A in PBC Library as the type of pairing parameters. The group order r is set as 160 bits long and the order of the base field q is set as 512 bits long. The number of bits in one component n_{bit} is set as 32, as the distance in the dataset can be represented within 7 decimal significant figures. We also assume that n_{bit} is a constant in the following part. We use the GNU Multiple Precision arithmetic library (GNU MP) to complete the bignum calculation.

In the evaluation of mobile encryption performance, the encryption procedure was implemented by Golang and bn256, an out-of-the-box pairing-based cryptography library. It enables the program to perform element mathematical operations and bilinear mapping. n_{bit} is also set as 32 in mobile encryption performance evaluation.

The procedure of mobile encryption evaluation is written by Golang. We use Gobind library to make it possible to call the Golang procedure from Objective-C. The evaluation of mobile encryption performance was conducted on iPhone 11 equipped with Apple A13 Bionic and iOS 14.3. In the other parts of this section, the proposed system is written by C programming language, and the experiments were done on a computer equipped with Intel Core i7-9700K of 3.6GHz and 64GB memory.

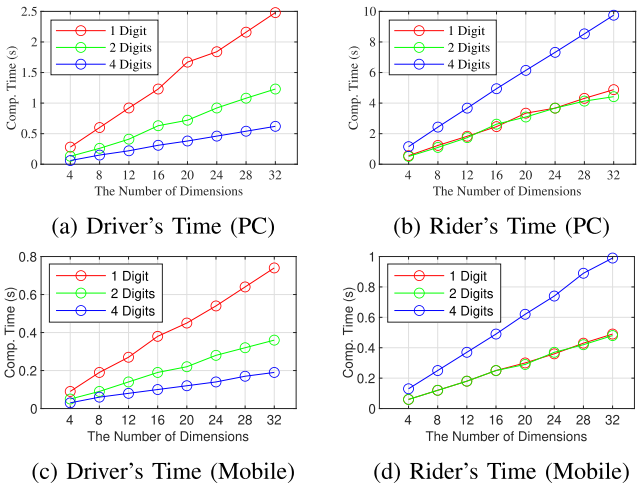
A. Datasets

The dataset [26] used for our experiment includes California's road network information. It contains 21048 road nodes and 21693 road segments(edges). They form a graph that represents the road network. We first calculate the shortest distance matrix which includes the shortest distances between each pair of nodes by Floyd-Warshall Shortest Path Algorithm [27]. The result is used for embedded vector generation of each road node and accuracy test. The road nodes' location vectors are public information. We locate all the drivers and riders randomly on the edges.

B. Accuracy

The core of ORH system is to match the closest driver for each rider. As the distance calculated by our proposed system is an approximate value, the matching result may be inaccurate. To evaluate the accuracy of our system, we generate 100 riders, get the nearest driver for each rider by our PPH-based ORH system, and count the number of riders whose the matched driver is the closest drivers calculated by the Shortest Distance Matrix mentioned in Sec. VI-A. We test our proposed system on different number of components. For each test, the experiment result is the mean of 5 trials.

As shown in Fig. 6a, our scheme based on RNE and PPH shows its significant performance. The accuracy rate increases with the number of components because the higher-dimensional vector includes more location information. Our evaluation result shows that the higher-dimension location vector can guarantee a higher accuracy rate. The accuracy rate could be above 90% when the dimension is higher than 20. In the test case with 32-dimension location vectors, the accuracy rate can reach almost 100%.



(a) Driver's Time (PC) (b) Rider's Time (PC)

(c) Driver's Time (Mobile) (d) Rider's Time (Mobile)

Fig. 5. The Comp. time of both driver and rider.

C. Performance

In this section, we evaluate the performance of the PPH-based ORH System in Communication Cost (Comm.) and Computation Cost (Comp.), i.e., the amount of data in network communications and computing time. Unless otherwise specified, *PPH-based ORH System* is the optimized scheme. Each experiment result is the mean of 10 cases.

The Performance of Driver: For a driver, the overhead is related to the block size and the number of vector's components. To better evaluate the cost of a driver, we generate the location vectors of different number of components (from 4 components to 32 components) and then divide each components into the blocks of different sizes (1 bits, 2 bits and 4 bits). The results in Fig. 5a and Fig. 5c demonstrate that the computation cost raises linearly as the dimension increases, and the cost decreases as the block size increases, since the larger block size leads to less blocks to be encrypted. Specifically, Table IV shows the computation and communication cost of a driver on PC and the block size is set as 2. Both communication cost and computation cost are positively correlated with the components. In 32 components, the communication cost is 136 KBs and the computation cost is 1.23 seconds. The communication cost and computation cost are quite acceptable.

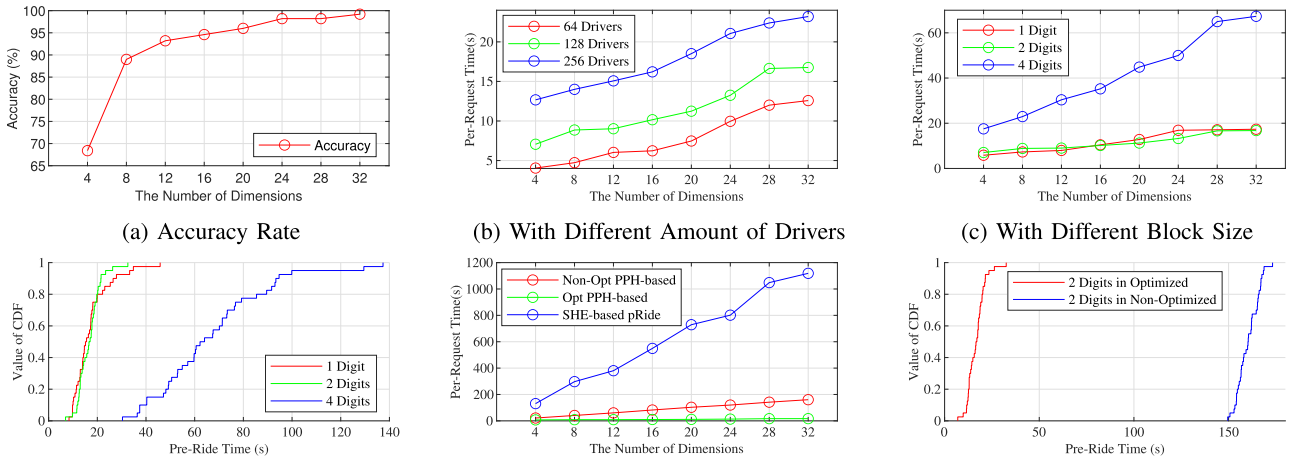
The Performance of Rider: Similarly, for a rider, the overhead also depends on the block size and the number of vector's components. Therefore, we evaluate the rider's cost by the same tests in driver's evaluation. Fig. 5b and Fig. 5d show that the computation cost also raises linearly as the dimension increases. However, unlike the results in the driver's evaluation, the cost increases as the block size increases, due

TABLE IV
PRE-RIDE PERFORMANCE OF SHE-BASED PRIDE AND PPH-BASED ORH SYSTEM WITH 128 DRIVERS AND BLOCK WITH 2 DIGITS

Number of Components	SHE-based pRide						PPH-based ORH System					
	Rider		Driver		Server		Rider		Driver		Server	
	Comm. (KBs)	Comp. (Secs)	Comm. (KBs)	Comp. (Secs)	Comm. (MBs)	Comp. (Secs)	Comm. (KBs)	Comp. (Secs)	Comm. (KBs)	Comp. (Secs)	Comm. (MBs)	Comp. (Secs)
4	2.0	3.4	2.0	3.4	8.7	130.7	69	0.51	17	0.13	0.00	7.06
8	4.1	6.9	4.1	6.9	17.2	297.3	138	1.1	34	0.26	0.00	8.86
12	6.1	10.3	6.1	10.2	25.8	380.3	207	1.74	51	0.41	0.00	9.02
16	8.2	13.8	8.2	13.8	34.3	549.4	276	2.62	68	0.63	0.00	10.16
20	10.3	17.0	10.3	17.0	42.9	729.1	345	3.09	85	0.72	0.00	11.24
24	12.3	20.6	12.3	20.6	51.4	801.4	414	3.67	102	0.92	0.00	13.25
28	14.4	24.0	14.4	24.1	59.9	1048.3	483	4.13	119	1.08	0.00	16.64
32	16.5	27.5	16.5	27.5	68.5	1118.7	552	4.42	136	1.23	0.00	16.76

to the increase in the number of values. A rider encrypts the numbers from 0 to $2^l - 1$ for each block instead of the block value itself in the driver's encryption procedure. For a dimension value, the number of values to be encrypted is $n \cdot 2^l$, where $n = \frac{32}{l}$ is the number of blocks. $\frac{32}{l}$ decreases with the increase of block size l, but 2^l increases exponentially at the same time. However, it is noteworthy that the computation cost with block size 1 and the cost with block size 2 are approximately equal, since the results for $\frac{32}{l} \cdot 2^l$ with $l = 1$ is equal to that with $l = 2$. Similar with the evaluation of driver, Table IV demonstrates the rider's performance with block size 2. As shown in Table IV, when conducting the evaluation on PC with 32 components, the rider's communication cost is 552 KBs and the computation cost is 4.42 seconds. The results in Table IV and Fig. 5 show that a rider's computation cost and communication cost are larger than a driver's, since a rider needs to encrypt and upload more values than a driver. However, a driver has to encrypt and upload the location more frequently, while a rider only needs to encrypt and upload the location when this rider wants to use ORH service. The trade-off is acceptable. Since bn256 uses a asymmetric curve, the computation costs on group G_1 is greater than the result on G_2. Thus, when using a mobile platform, the cost of rider is greater than that of driver when the block size is 1 digit. Nonetheless, the driver's cost is still greater than rider's if the block size is larger, e.g. 2 digits and 4 digits, as the same as the results on PC.

The Performance of ORH Server: In the initialization procedure, the ORH server calculates the shortest distance matrix and generates the RNE vectors for all the nodes. The Floyd-Warshall algorithm in multi-threading mode costs 52 minutes to generate the shortest distance matrix. In the case of 32 components, the ORH server spends 62 seconds on generating the RNE vectors for all 21048 nodes. The overhead of ORH server is related to three aspects, i.e., the number of available drivers, the number of components and the block size. In our experiment, we generate the location vectors of different number of components. Then we record the processing time for different number of available drivers, different block sizes, and different matching schemes. The optimized proposed system's computation cost is related to the data, as the distance calculation may be terminated early because the current driver is impossible to be the most suitable driver according to Sec. IV-D. We use *Cumulative Distribution Function*(CDF) Graph to observe the distribution of the results with different block sizes, and the results with or without optimization. Fig. 6 shows the results in our experiments. According to Fig. 6b, we observe that the computation cost increases as the number of available drivers becomes large. This is due to the fact that the ORH server calculates the distance between one rider and each driver. Similar with the evaluation of rider and driver, we try to find the relationship between the computation overhead and block size. As shown in Fig. 6c, for the same reason we described in the rider's evaluation, the computation overhead with block size 4 increases compared with the cost with block size 1 and 2, while the cost with block size 1 is equal to that with block size 2. Fig. 6d illustrates the distribution of the results in 32 components in Fig. 6c. As shown in Fig. 6d, we observe that about 50% results are smaller than about 15 secs with block size 1 and block size 2, and smaller than about 60 secs with block size 4, as the same as the results shown in Fig. 6c. The experiment results are distributed evenly. Table IV also shows the performance of ORH server with 128 available drivers and block size 2, and the performance of SHE-based pRide. Fig. 6e demonstrates our experiment results in Table IV and the performance of the non-optimized PPH-based ORH system under the same conditions. As shown in Table IV and Fig. 6e, both non-optimized system and optimized system decrease the computation cost significantly compared with SHE-based pRide. The optimized PPH-based ORH system's computation overhead is generally 95% less than SHE-based pRide. To better evaluate the computation cost of our non-optimized system and optimized system, Fig. 6f depicts the distribution of results with block size 2. About 50% results are smaller than about 170 seconds in the non-optimized system, and smaller than about 20 seconds in optimized system. The evaluation results confirm that our optimized design improves the efficiency significantly. In addition, the matching procedure could be implemented



(a) Accuracy Rate (b) With Different Amount of Drivers (c) With Different Block Size

(d) CDF with Different Block Size (e) With Different Scheme (f) CDF with Opt or Non-Opt

Fig. 6. Evaluation of system performance.

in a multi-threading setting in the actual environment. Thus, the index could be divided into several parts. Each CPU core could compute the RNE distances between the rider and a part of the drivers in the index. The ORH server could select the closest driver based on the distances from these cores. As there are usually many CPU cores in the server, the computation cost could be reduced in a multi-threading setting. We also observe that the communication cost is 0 since it does not need to contact any other entities in the matching procedure as shown in Table IV.

VII. Related Work

A. Privacy-Preserving Ride-Matching System

There exists a line of work that studies the problem of privacy-preserving ride-matching [28]–[31]. Finginal *et al.* [32] first considered the privacy issue in car-pooling systems and proposed to use obfuscated symbols to hide real locations. C. Bettini *et al.* [33] also utilized this non-cryptographic scheme to solve the problem of anonymity. In [34], the authors devised a cloaking technique to hide the user location. Similarly, P. Goel *et al.* [35] proposed an obfuscate scheme to obfuscate the location information. However, all these non-cryptographic solutions would incur the loss of accuracy because they leverage imprecise locations for distance calculation. Y. Duan *et al.* [36] studied the loss of the matching performance in cloaking scheme and proposed a pricing scheme to make up for the individual loss.

To improve ride-matching accuracy, some cryptographic solutions have been proposed based on different cryptographic primitives. PrivateRide [5] is the first system to support encrypted ride-matching, but they only provide limited security guarantees for riders. In [6], Pham *et al.* proposed to use Somewhat Homomorphic Encryption (SHE) [9] to improve the security guarantee. Wang *et al.* [37] devised a dynamic secure spatial query scheme based on Euclidean Distance. However, Euclidean Distance is not accurate in practical ride-matching systems because vehicles need to be driven along the roads. P. Hallgren *et al.* proposed PrivatePool [38], a decentralized private-preserving ride-sharing scheme based on secure multi-party computation. In this scenerio, both the rider and the driver have their own trip paths, and the driver wants to take a rider with a similar trip path to reduce the cost. Thus, PrivatePool matches the rider and the driver according to the similarity of their own planned trip paths, i.e., how far the respective starting & ending points are, or how large of an overlap there is between two trip paths. In [39], E. Pagnin *et al.* proposed a decentralized time-aware privacy-preserving ridesharing scheme. In contrast to PrivatePool, it improves matching performance and supports time preference. A recent work called pRide [7] has been proposed that can securely make ride-matching based on the road distance between rider's start point and driver's current location by using RNE and Yao's GCs [40]. However, pRide assumes a non-colluding third-party server who holds the decryption key for distance calculation. In the follow-up design [8], they also proposed to leverage trusted SGX to improve the matching efficiency, but its security assumption still relies on trusted hardware at the server-side.

B. Secure Distance Calculation and Comparison

Our proposed research is also related with the branch of work on secure distance calculation [41]–[45]. In [46] and [47], the authors proposed a secure hamming distance calculation scheme based on Oblivious Transfer. However, it required multiple interactions among the participants so that it was subject to high bandwidth overhead. Aly *et al.* [48] introduced the protocols to solve the shortest path problem and the maximum flow problem by using multi-party computing, but it leads to high computation cost. Therefore it will be impratical if deals with large graph. Cao *et al.* [49] proposed a privacy-preserving similarity measure scheme over encrypted data. In this scheme, a vector is used to described one document index and inner product of the two vectors is used to measure the similarity of two documents. In [50], Xi *et al.* utilized Private Information Retrieval (PIE) scheme [51] and Garble Circuits to handle the location privacy problem in

mobile navigation without the leakage of the starting point and the destination. In [52], Meng *et al.* proposed an approximate distance query scheme based on Distance Oracle [53] and Somewhat Homomorphic Encryption [9]. Shen *et al.* [54] proposed an approximate shortest distance query scheme which is based on Somewhat Homomorphic Encryption and tree-based ciphertexts comparison protocol. Wang *et al.* [55] proposed SecGDB, the exact shortest distance query scheme based on Dijkstra shortest path algorithm, Paillier Cryptosystem [56] and Garble Circuit [11]. However, the schemes based on homomorphic encryption are not efficient enough. They could not provide the acceptable calculation time for ORH system due to its complicated geographical information.

VIII. Conclusion

In this paper, we studied the online ride-hailing service schemes and their security designs, and proposed a privacy-preserving online ride-hailing system without involving a third trusted party. Our proposed system can match the nearest driver for each rider while preserving both driver privacy and rider privacy without a trusted third-party's assistance. In our system, we redesigned Privacy-Preserving Hash (PPH) to construct a secure difference computing module, which can calculate the difference between two values safely. By using Road Network Embedding with our secure difference computing module, the proposed system can calculate the distance between rider and one driver over the encrypted data. We analyzed the security of our proposed system. The evaluation results based on real-world datasets showed our proposed system's high matching accuracy and efficiency. As future work, we plan to explore advanced road network matching schemes to support dynamic road weights, which consider real road length and the current congestion levels. Meanwhile, we leave how to detect malicious riders/drivers who intentionally leak their private keys as our future work.

References

[1] (2020). *Uber*. [Online]. Available: https://www.uber.com/

[2] (2020). *Didi*. [Online]. Available: https://www.didiglobal.com/

[3] (2020). *GrabTaxi*. [Online]. Available: https://www.grab.com/

[4] (2020). *Lyft*. [Online]. Available: https://www.lyft.com/

[5] A. Pham *et al.*, "PrivateRide: A privacy-enhanced ride-hailing service," *Proc. Privacy Enhancing Technol.*, vol. 2017, no. 2, pp. 38–56, Apr. 2017.

[6] A. Pham, I. Dacosta, G. Endignoux, J. R. T. Pastoriza, K. Huguenin, and J.-P. Hubaux, "ORide: A privacy-preserving yet accountable ride-hailing service," in *Proc. 26th USENIX Secur. Symp. (USENIX Secur.)*, 2017, pp. 1235–1252.

[7] Y. Luo, X. Jia, S. Fu, and M. Xu, "PRide: Privacy-preserving ride matching over road networks for online ride-hailing service," *IEEE Trans. Inf. Forensics Security*, vol. 14, no. 7, pp. 1791–1802, Jul. 2019.

[8] Y. Luo, X. Jia, H. Duan, C. Wang, M. Xu, and S. Fu, "PRide: Private ride request for online ride hailing service with secure hardware enclave," in *Proc. Int. Symp. Qual. Service*, Jun. 2019, pp. 1–10.

[9] J. Fan and F. Vercauteren, "Somewhat practical fully homomorphic encryption," *IACR Cryptol. ePrint Arch.*, vol. 2012, p. 144, Mar. 2012.

[10] C. Shahabi, M. R. Kolahdouzan, and M. Sharifzadeh, "A road network embedding technique for k-nearest neighbor search in moving object databases," *Geoinformatica*, vol. 7, no. 3, pp. 255–273, 2015.

[11] Y. Huang, D. Evans, J. Katz, and L. Malka, "Faster secure two-party computation using garbled circuits," in *Proc. USENIX Secur. Symp.*, 2011, vol. 201, no. 1, pp. 331–335.

[12] V. Costan and S. Devadas, "Intel SGX explained," *IACR Cryptol. ePrint Arch.*, vol. 2016, no. 86, pp. 1–118, 2016.

[13] D. Cash, F.-H. Liu, A. O'Neill, M. Zhandry, and C. Zhang, "Parameter-hiding order revealing encryption," in *Proc. Int. Conf. Theory Appl. Cryptol. Inf. Secur.* Cham, Switzerland: Springer, 2018, pp. 181–210.

[14] O. Pandey and Y. Rouselakis, "Property preserving symmetric encryption," in *Proc. Annu. Int. Conf. Theory Appl. Cryptograph. Techn.* Berlin, Germany: Springer, 2012, pp. 375–391.

[15] R. Agrawal, J. Kiernan, R. Srikant, and Y. Xu, "Order preserving encryption for numeric data," in *Proc. ACM SIGMOD Int. Conf. Manage. Data (SIGMOD)*, 2004, pp. 563–574.

[16] A. Boldyreva, N. Chenette, Y. Lee, and A. O'neill, "Order-preserving symmetric encryption," in *Proc. Annu. Int. Conf. Theory Appl. Cryptograph. Techn.* Berlin, Germany: Springer, 2009, pp. 224–241.

[17] A. Boldyreva, N. Chenette, and A. O'Neill, "Order-preserving encryption revisited: Improved security analysis and alternative solutions," in *Proc. Annu. Cryptol. Conf.* Berlin, Germany: Springer, 2011, pp. 578–595.

[18] F. Kerschbaum, "Frequency-hiding order-preserving encryption," in *Proc. 22nd ACM SIGSAC Conf. Comput. Commun. Secur.*, Oct. 2015, pp. 656–667.

[19] Y. Guo, X. Yuan, X. Wang, C. Wang, B. Li, and X. Jia, "Enabling encrypted rich queries in distributed key-value stores," *IEEE Trans. Parallel Distrib. Syst.*, vol. 30, no. 6, pp. 1283–1297, Jun. 2019.

[20] X. Yuan, Y. Guo, X. Wang, C. Wang, B. Li, and X. Jia, "EncKV: An encrypted key-value store with rich queries," in *Proc. ACM Asia Conf. Comput. Commun. Secur.*, Apr. 2017, pp. 423–435.

[21] (2020). *AWS Key Management Service Documentation*. [Online]. Available: https://docs.aws.amazon.com/kms/

[22] (2021). *IBM Cloud Key Protect*. [Online]. Available: https://cloud.ibm.com/docs/key-protect?topic=key-protect-key-rotation

[23] R. Curtmola, J. Garay, S. Kamara, and R. Ostrovsky, "Searchable symmetric encryption: Improved definitions and efficient constructions," in *Proc. 13th ACM Conf. Comput. Commun. Secur. (CCS)*, 2006, pp. 895–934.

[24] D. Cash, F.-H. Liu, A. O'Neill, and C. Zhang, "Reducing the leakage in practical order-revealing encryption," *IACR Cryptol. ePrint Arch.*, vol. 2016, no. 661, p. 661, 2016.

[25] B. Lynn, "On the implementation of pairing-based cryptosystems," Ph.D. dissertation, Dept. Comput. Sci., Stanford Univ., Stanford, CA, USA, 2007.

[26] F. Li, D. Cheng, M. Hadjieleftheriou, G. Kollios, and S.-H. Teng, "On trip planning queries in spatial databases," in *Proc. Int. Symp. Spatial Temporal Databases*. Berlin, Germany: Springer, 2005, pp. 273–290.

[27] R. W. Floyd, "Algorithm 97: Shortest path," *Commun. ACM*, vol. 5, no. 6, p. 345, Jun. 1962.

[28] A. B. T. Sherif, K. Rabieh, M. M. E. A. Mahmoud, and X. Liang, "Privacy-preserving ride sharing scheme for autonomous vehicles in big data era," *IEEE Internet Things J.*, vol. 4, no. 2, pp. 611–618, Apr. 2017.

[29] Y. Khazbak, J. Fan, S. Zhu, and G. Cao, "Preserving location privacy in ride-hailing service," in *Proc. IEEE Conf. Commun. Netw. Secur. (CNS)*, May 2018, pp. 1–9.

[30] C. Cao and X. Zhu, "Practical secure transaction for privacy-preserving ride-hailing services," *Secur. Commun. Netw.*, vol. 2018, pp. 1–8, Feb. 2018.

[31] U. M. Aïvodji, K. Huguenin, M.-J. Huguet, and M.-O. Killijian, "SRide: A privacy-preserving ridesharing system," in *Proc. 11th ACM Conf. Secur. Privacy Wireless Mobile Netw.*, Jun. 2018, pp. 40–50.

[32] J. Friginal, S. Gambs, J. Guiochet, and M.-O. Killijian, "Towards privacy-driven design of a dynamic carpooling system," *Pervas. Mobile Comput.*, vol. 14, pp. 71–82, Oct. 2014.

[33] C. Bettini, S. Mascetti, X. S. Wang, D. Freni, and S. Jajodia, "Anonymity and historical-anonymity in location-based services," in *Privacy in Location-Based Applications*. Berlin, Germany: Springer, 2009, pp. 1–30.

[34] C.-Y. Chow, M. F. Mokbel, and X. Liu, "A peer-to-peer spatial cloaking algorithm for anonymous location-based service," in *Proc. 14th Annu. ACM Int. Symp. Adv. Geographic Inf. Syst. (GIS)*, 2006, pp. 171–178.

[35] P. Goel, L. Kulik, and K. Ramamohanarao, "Privacy-aware dynamic ride sharing," *ACM Trans. Spatial Algorithms Syst.*, vol. 2, no. 1, pp. 1–41, Apr. 2016.

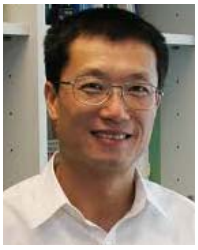
[36] Y. Duan, G. Gao, M. Xiao, and J. Wu, "A privacy-preserving order dispatch scheme for ride-hailing services," in *Proc. IEEE 16th Int. Conf. Mobile Ad Hoc Sensor Syst. (MASS)*, Nov. 2019, pp. 118–126.

[37] F. Wang *et al.*, "Efficient and privacy-preserving dynamic spatial query scheme for ride-hailing services," *IEEE Trans. Veh. Technol.*, vol. 67, no. 11, pp. 11084–11097, Nov. 2018.

[38] P. Hallgren, C. Orlandi, and A. Sabelfeld, "PrivatePool: Privacy-preserving ridesharing," in *Proc. IEEE 30th Comput. Secur. Found. Symp. (CSF)*, Aug. 2017, pp. 276–291.
[39] E. Pagnin, G. Gunnarsson, P. Talebi, C. Orlandi, and A. Sabelfeld, "TOPPool: Time-aware optimized privacy-preserving ridesharing," *Proc. Privacy Enhancing Technol.*, vol. 2019, no. 4, pp. 93–111, 2019.
[40] A. C. Yao, "Protocols for secure computations," in *Proc. 23rd Annu. Symp. Found. Comput. Sci. (SFCS)*, Nov. 1982, pp. 160–164.
[41] W. K. Wong, D. W.-L. Cheung, B. Kao, and N. Mamoulis, "Secure kNN computation on encrypted databases," in *Proc. 35th SIGMOD Int. Conf. Manage. Data (SIGMOD)*, 2009, pp. 139–152.
[42] K. Mouratidis and M. L. Yiu, "Shortest path computation with no information leakage," 2012, *arXiv:1204.6076*. [Online]. Available: http://arxiv.org/abs/1204.6076
[43] B. K. Samanthula, F.-Y. Rao, E. Bertino, and X. Yi, "Privacy-preserving protocols for shortest path discovery over outsourced encrypted graph data," in *Proc. IEEE Int. Conf. Inf. Reuse Integr.*, Aug. 2015, pp. 427–434.
[44] D. Xie *et al.*, "Practical private shortest path computation based on oblivious storage," in *Proc. IEEE 32nd Int. Conf. Data Eng. (ICDE)*, May 2016, pp. 361–372.
[45] H. Zhu, B. Wu, M. Xie, Z. Cui, and Z. Wu, "Secure shortest path search over encrypted graph supporting synonym query in cloud computing," in *Proc. IEEE Trustcom/BigDataSE/ISPA*, Aug. 2016, pp. 497–504.
[46] J. Bringer, H. Chabanne, and A. Patey, "SHADE: Secure Hamming distance computation from oblivious transfer," in *Proc. Int. Conf. Financial Cryptogr. Data Secur.* Berlin, Germany: Springer, 2013, pp. 164–176.
[47] J. Bringer, H. Chabanne, M. Favre, A. Patey, T. Schneider, and M. Zohner, "GSHADE: Faster privacy-preserving distance computation and biometric identification," in *Proc. 2nd ACM Workshop Inf. Hiding Multimedia Secur. (IH MMSec)*, 2014, pp. 187–198.
[48] A. Aly, E. Cuvelier, S. Mawet, O. Pereira, and M. Van Vyve, "Securely solving simple combinatorial graph problems," in *Proc. Int. Conf. Financial Cryptogr. Data Secur.* Berlin, Germany: Springer, 2013, pp. 239–257.
[49] N. Cao, Z. Yang, C. Wang, K. Ren, and W. Lou, "Privacy-preserving query over encrypted graph-structured data in cloud computing," in *Proc. 31st Int. Conf. Distrib. Comput. Syst.*, Jun. 2011, pp. 393–402.
[50] Y. Xi, L. Schwiebert, and W. Shi, "Privacy preserving shortest path routing with an application to navigation," *Pervas. Mobile Comput.*, vol. 13, pp. 142–149, Aug. 2014.
[51] B. Chor, O. Goldreich, E. Kushilevitz, and M. Sudan, "Private information retrieval," in *Proc. IEEE 36th Annu. Found. Comput. Sci.*, Oct. 1995, pp. 41–50.
[52] X. Meng, S. Kamara, K. Nissim, and G. Kollios, "GRECS: Graph encryption for approximate shortest distance queries," in *Proc. 22nd ACM SIGSAC Conf. Comput. Commun. Secur.*, Oct. 2015, pp. 504–517.
[53] A. D. Sarma, S. Gollapudi, M. Najork, and R. Panigrahy, "A sketch-based distance oracle for Web-scale graphs," in *Proc. 3rd ACM Int. Conf. Web Search Data Mining (WSDM)*, 2010, pp. 401–410.
[54] M. Shen, B. Ma, L. Zhu, R. Mijumbi, X. Du, and J. Hu, "Cloud-based approximate constrained shortest distance queries over encrypted graphs with privacy protection," *IEEE Trans. Inf. Forensics Security*, vol. 13, no. 4, pp. 940–953, Apr. 2018.
[55] Q. Wang, K. Ren, M. Du, Q. Li, and A. Mohaisen, "SecGDB: Graph encryption for exact shortest distance queries with efficient updates," in *Proc. Int. Conf. Financial Cryptogr. Data Secur.* Cham, Switzerland: Springer, 2017, pp. 79–97.
[56] P. Paillier, "Public-key cryptosystems based on composite degree residuosity classes," in *Proc. Int. Conf. Theory Appl. Cryptograph. Techn.* Berlin, Germany: Springer, 1999, pp. 223–238.

Hongcheng Xie (Graduate Student Member, IEEE) received the B.Eng. degree in software engineering from the University of Electronic Science and Technology of China in 2019. He is currently pursuing the Ph.D. degree with the Department of Computer Science, City University of Hong Kong. His research interests include information security on cloud computing and blockchain.

Yu Guo (Member, IEEE) received the B.E. degree in software engineering from Northeastern University in 2013, and the M.Sc. degree in electronic commerce and the Ph.D. degree in computer science from the City University of Hong Kong, in 2014 and 2019, respectively. He has been a Post-Doctoral Researcher and a Research Fellow with the City University of Hong Kong. He is currently a Lecturer with the School of Artificial Intelligence, Beijing Normal University. His research interests include cloud computing security, network security, privacy-preserving data processing, and blockchain technology. He was a co-recipient of the Best Paper Award of MMM 2016 and IEEE ICDCS 2020.



Xiaohua Jia (Fellow, IEEE) received the B.Sc. and M.Eng. degrees from the University of Science and Technology of China, in 1984 and 1987, respectively, and the D.Sc. degree in information science from The University of Tokyo in 1991. He is currently the Chair Professor with the Department of Computer Science, City University of Hong Kong. His research interests include cloud computing and distributed systems, computer networks, wireless sensor networks, and mobile wireless networks. He is fellow of the IEEE Computer Society. He is an Editor of the IEEE TRANSACTIONS ON PARALLEL AND DISTRIBUTED SYSTEMS from 2006 to 2009, *Wireless Networks*, *World Wide Web* Journal, and *Journal of Combinatorial Optimization*. He is the General Chair of ACM MobiHoc 2008, a TPC Co-Chair of IEEE MASS 2009, IEEE GlobeCom 2010-Ad Hoc, and Sensor Networking Symposium, an Area-Chair of IEEE INFOCOM 2010, and a Panel Co-Chair of IEEE INFOCOM 2011.