Nereus: Anonymous and Secure Ride-Hailing Service Based on Private Smart Contracts

Meng Li, *Member, IEEE*, Yifei Chen, Chhagan Lal, *Member, IEEE*, Mauro Conti, *Fellow, IEEE*, Fabio Martinelli, and Mamoun Alazab, *Senior Member, IEEE*

Abstract—Security and privacy issues have become a major hindrance to the broad adoption of Ride-Hailing Services (RHSs). In this article, we introduce a new collusion attack initiated by the Ride-Hailing Service Provider (RHSP) and a driver that could easily link the real riders and their anonymous requests (credentials). Besides this attack, existing work requires heavy computations to execute user matching, and it is challenging for riders to verify matching results. Meanwhile, a malicious driver may cancel an assigned ride order due to its short distance. To address these issues, we present a RHS system named Nereus to support collusion resistance, efficiency, verifiability, and accountability. First, we integrate a smart contract into a Software Guard Extensions (SGX) enclave to establish a *private smart contract* for collusion resistance. We use a Bloom filter to achieve efficient matching. Second, we leverage privacy-preserving range query and Merkle proofs to make matching results verifiable. Meanwhile, we adopt short group signatures to provide anonymous authentication and deposit commitments to hold the runaway driver accountable. We formally state and prove the security and privacy of Nereus. We build a prototype based on Ethereum and SGX to conduct extensive performance analysis in regard to gas costs, computational costs, and communication overhead. Experimental results show that Nereus significantly improves over existing schemes in terms of computational costs.

Index Terms—Ride-hailing services, privacy, collusion attack, private smart contract, SGX

1 Introduction

The development of popular Ride-Hailing Services (RHSs) is spawned by mature cloud computing, wireless communication, and sensor-enriched smartphones. RHSs have been prospering as a global phenomenon over the past decade and offering users (riders and drivers) many benefits.

During ride-hailing, user matching is the most important component because it is related to rider's waiting time, driver's income, system efficiency, and the RHSP's profits. To form a ride, the rider and the driver submit ride data to the Ride-Hailing Service Provider (RHSP) for user matching. These data include identity, current location, destination, and requirements, e.g., vehicle brand, driver experience, and driver reputation. Such information correlates with sensitive information about the user, e.g. home, workplace, and propensities. Besides, the RHSP is considered an untrustworthy entity due to cyber attacks or mischievous employees. One report showed that 2.7 million Uber users in the UK were affected by a mass data breach in 2016 [1]. Hence, sharing ride data with the RHSP raises many significant security and privacy concerns [2], [3], [4], [5], [6], [7]. Without careful design, these concerns will hobble to what proves to be a more convenient mode of transportation. In the literature, there are multiple schemes utilizing cryptographic primitives and data structures to achieve privacy-preserving RHSs (e.g., ORide [8], DAP-DAD [9], pRide [10], FICA [11], CoRide [12], B-Ride [13]).

Our *motivation* originates from a new collusion attack from the RHSP and a driver to violate a rider's identity privacy. As depicted in Fig. 1, an anonymous rider Alice and a driver Bob are matched. Given Alice's anonymous request, the RHSP matches her to a colluding driver Bob. When Alice is aboard Bob's vehicle, Bob sees Alice's face and location (i.e., a kind of identity). Next, Bob and the RHSP can associate the information with Alice's anonymous request. Assuming that the RHSP continues to collude with a group of drivers to track Alice's future requests, the colluding parties can link the anonymous requests (from the digital world) and Alice (from the physical world). Consequently, the RHSP acquires the requesting time, pick-up location, destination, and trajectories

- *Meng Li is with the Key Laboratory of Knowledge Engineering with Big Data (Hefei University of Technology), Ministry of Education, Hefei, Anhui 230002, China, and with the School of Computer Science and Information Engineering, Hefei University of Technology Anhui Province Key Laboratory of Industry Safety and Emergency Technology, Hefei 230002, China, and also with the Intelligent Interconnected Systems Laboratory of Anhui Province (Hefei University of Technology), Hefei, Anhui 230002, China. E-mail: mengli@hfut.edu.cn.*
- *Yifei Chen is with Solution and Architecture Research Department, NSFOCUS, Nanjing, Jiangsu 211106, China. E-mail: chenyifei@nsfocus.com.*
- *Chhagan Lal is with the Department of Intelligent Systems, CyberSecurity Group, TU Delft, 2628 Delft, Netherlands. E-mail: c.lal@tudelft.nl.*
- *Mauro Conti is with the Department of Mathematics and HIT Center, University of Padua, 35131 Padua, Italy, and also with the Department of Intelligent Systems CyberSecurity Group, TU Delft, 2628 Delft, Netherlands. E-mail: conti@math.unipd.it.*
- *Fabio Martinelli is with the Institute for Informatics and Telematics, National Research Council of Italy (CNR), 56123 Pisa, Italy. E-mail: fabio.martinelli@iit.cnr.it.*
- *Mamoun Alazab is with the College of Engineering, IT and Environment, Charles Darwin University, Casuarina, NT 0810, Australia. E-mail: alazab.m@ieee.org.*

Manuscript received 16 March 2022; revised 17 June 2022; accepted 16 July 2022. Date of publication 19 July 2022; date of current version 11 July 2023. The work was supported by the National Natural Science Foundation of China (NSFC) under Grant 62002094, in part by Anhui Provincial Natural Science Foundation under Grant 2008085MF196, and in part by EU LOCARD Project under Grant H2020-SU-SEC-2018-832735, and EU project E-CORRIDOR. (Corresponding author: Yifei Chen.)
Digital Object Identifier no. 10.1109/TDSC.2022.3192367

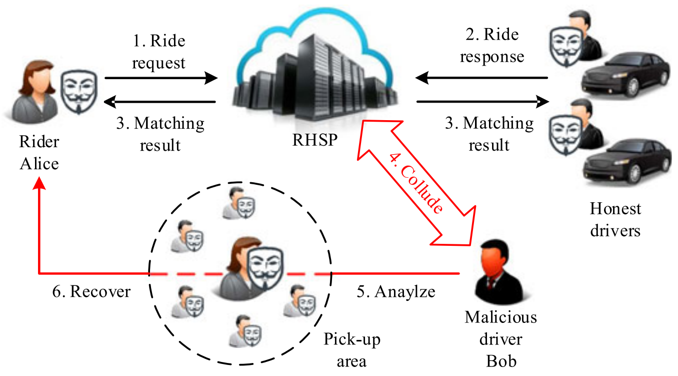


Fig. 1. Collusion attack between RHSP and driver.

of Alice. All this information eventually leads to identity recognition. In reality, reports say that the Cyberspace Administration of China found “serious violations” in how Didi used personal information [14] and Uber employees use “God View” to track customers in real time [15], highlighting identity privacy. Such unfortunate news calls for anonymous and secure RHSs. We agree that hiding a rider’s identity cannot fundamentally prevent driver crimes. This requires in-depth cooperation between ride-hailing companies, labor union, and police department. We aim to protect riders’ identity privacy under collusion attack to further disassociate anonymous requests from riders. In this way, we eliminate (to some extent) the possible malicious intentions in their initial state when the colluding parties pry into the request-identity association.

To solve the above-mentioned problem, several techniques seem to offer a solution. First, profile-matching [16], a well-studied technique, enables an initiating user to find from nearby candidates the one whose profile best matches with hers/his while not disclosing users’ sensitive information [17], [18]. Most approaches adopt secure multiparty computation [16], secret sharing [16], public encryption [17] to privacy-preserving matching. However, they either suffer from too much computation and multiple rounds of communication [17] or rely on multiple servers [18] to complete the matching [18]. Second, in a mix-net [19], one mixer (or several mixers) anonymizes its inputs by eliminating any observable connection between the inputs and final outputs. We can use a mixer, e.g., a road-side unit (RSU) or another cloud server, to blend riders’ requests before the matching begins, but it remains a challenge to ask the RHSP to return the matching results to the associated riders. If we consider the mixer as an honest party in charge of recording the blending operations or just matching users locally as a fog/edge node [4], then the security assumption is too strong for existing RHSs.

The above-mentioned obstacles lead to the first technical challenge in designing a privacy-preserving ride-hailing scheme. *Challenge I*: *how to defend against collusion attack and protect identity privacy for riders*. It is challenging to preserve the identity privacy of riders under collusion attack between RHSP and driver. This is because the RHSP controls the user matching process and knows all the matching relations between riders and drivers. Meanwhile, the colluding driver who receives a matching result from the process is just what RHSP needs to link the rider’s request to the rider. We initiate preventing the colluding parties (i.e., driver and RHSP) from knowing which anonymous rider corresponds to the actual rider picked up by the malicious driver. In contrast to the centralized service model, blockchain [20] is a distributed, immutable, and verifiable ledger that enables transactions between distrusting users without relying on a centralized entity. Smart contract (SC) [21] allows automatic execution of predefined codes on top of a blockchain. Blockchain and SCs can be used to address the problem of a single point of failure for the RHSP and autonomous user matching. Meanwhile, Software Guard eXtensions (SGX) [5], [22], [23], [24] add support for data confidentiality and integrity with secure enclaves. SGX enables an isolated and secure space that sheds light on a new approach to perform user matching. These techniques motivate us to build a self-executed environment secured by SGX to carry out collusion-resistant user matching.

Besides Challenge I, we face three technical challenges in designing a secure ride-hailing scheme. *Challenge II*: *how to reduce the computation costs for RHS entities*. Most privacy-preserving RHS schemes rely on heavy cryptography to process requests and responses. It is challenging further to slash down the computation costs under the current infrastructure. *Challenge III*: *how to guarantee matching verifiability for users*. The RHSP or an RSU conducts the traditional matching process. Such opacity nevertheless causes a loss of control from users on their matching results. It is challenging for a user to verify the received matching result, i.e., whether it is valid. *Challenge IV*: *how to guarantee accountability for users*. A user may actively deviate from the protocol, which corresponds to the malicious model [25]. For example, a malicious driver refuses to pick up the assigned rider due to a low fare after a short ride. Such a runaway driver should be punished. It is challenging to hold the runaway user accountable in such an anonymous platform. Some malicious users may send data that is not consistent with the one in ride request/response when meeting the matched user.

We propose an anonymous and secure RHS system based on Private Smart Contracts (PSCs) to address the aforementioned problems. We first establish the function objective, i.e., collusion resistance, efficiency, matching verifiability, and accountability. Next, we design two protocols to realize our goals. The basic protocol realizes collusion resistance and efficiency. The advanced protocol extends the basic protocol to achieve matching verifiability and accountability.

In the basic protocol, we construct a simple but efficient matching algorithm based on the Bloom filter. To realize distributed matching, we build the RHS scheme on top of a public blockchain. We note that a user matching smart contract (UMSC) [21], [26], [27] is designed and then deployed on RSUs to automatically pair riders with drivers. We further establish a secure execution environment based on Intel’s SGX [22], [23] to integrate the above UMSC and establish a PSC, while protecting the ride request, ride response, and the matching result. In the advanced protocol, we design a ride verifying smart contract (RVSC) stemming from fair exchange [28] to guarantee verifiability and accountability. A deposit commitment on user features and a “Proof-of-Runaway” (PoR) are designed to hold malicious users accountable. In summary, our contributions are as follows:

- The design of a new ride-hailing ecosystem based on a public blockchain and SGX-protected SCs, Nereus, which addresses the collusion attack, efficiency, matching verifiability, and accountability in RHS.
- The definition of privacy and two security levels, and the design of two corresponding protocols (basic and advanced) for Nereus to ensure security and privacy. The basic protocol is used to show how we defend against the collusion attack and the advanced protocol is designed to add more security features.
- The formal definitions and proofs of security and privacy. The implementation of Nereus and extensive performance analysis based on Ethereum and Intel SGX. We choose the two technologies for their wide adoption.

2 Background and Related Work

2.1 Blockchain and Smart Contract

Blockchain is a public, decentralized, and tamper-proof ledger that was initially used as an underlying technique to solve the double-spending problem in cryptocurrencies [20]. It integrates cryptography, economic modeling, peer-to-peer networking and decentralized consensus to achieve distributed database synchronization. SC is a piece of codes deployed on the blockchain with a unique address and state variables. It autonomously performs the functions triggered by specific transactions in a predetermined manner. Executing functions in SCs raises some execution fees to incentivize peers and mitigate denial of service attacks.

2.2 Indistinguishable Bloom Filter

An Indistinguishable Bloom Filter (IBF) contains an array $\mathcal{IBF}$ of l cell twins, z pseudo random hash functions $H_1, H_2, \ldots, H_z$, and a random oracle H_{z+1}. Each cell twin has two cells and each cell stores either '0' or '1'. H_{z+1} determine which cell stores '1'. In the beginning, the chosen cell is set to 0 and the unchosen cell is set to 1. An item it is hashed to z twin cells $\mathcal{IBF}[H_1(it)], \mathcal{IBF}[H_2(it)], \ldots, \mathcal{IBF}[H_z(it)]$. The z chosen cells are set to '1' and the unchosen cells are set to '0'. An IBF with m cell twins has m '1's and the chosen cell is randomly picked. Thus, any polynomial-time adversary distinguishes the chosen cell from the other cell correctly with a negligible probability over $1/2$. Furthermore, the IBF is adaptively secure using a random oracle model.

2.3 Intel SGX

Software Guard Extensions (SGX) is a hardware extension of Intel Architecture that enables an application to establish a protected container, i.e., an enclave. It is protected by the processor through access control. Accesses to the enclave from outside the enclave are denied [22]. Only the processor and programs in the enclave can access the code and data in plaintext. Such an enclave provides confidentiality and integrity. To protect the enclave outside of the CPU, it is encrypted and integrity-protected by a specific key. The code executed within an enclave runs in the context of the creating process. There are already many works on developing privacy-enhanced applications with SGX. VC3 [29] leveraged SGX processors to isolate memory regions on individual computers and secure distributed MapReduce computations. SDTE [30] established a secure execution environment atop SGX to protect the source data and the data analysis results in data trading. SecGrid [31] designed a secure SGX-enabled smart grid system to support rich functionalities on customers' private data.

2.4 Privacy-Preserving RHSs

Pham et al.[8] presented a privacy-preserving RHS scheme ORide to match riders with drivers without leaking users' identities or locations. ORide leveraged homomorphic encryption and optimizations for ciphertext packing and transformed processing. Sherif et al. [9] proposed a privacy-preserving ride sharing scheme including three sub-schemes. They leveraged a group signature scheme to ensure user anonymity and a similarity measurement technique to preserve trip data privacy. Luo et al. [10] presented a privacy-preserving RHS scheme pRide to prevent the leakage of the mobility patterns and protect the location privacy of drivers and riders. pRide used a road network embedding technique and cryptographic primitives to securely estimate the shortest distances between users approximately.

2.5 Blockchain-Based RHSs

A ride-hailing system based on blockchain has several advantages over a centralized one: (1) There is no single point of failure; (2) The ride-hailing transactions are auditable; and (3) it allows different ride-hailing companies to cooperate. Li et al. [11] present an efficient and privacy-preserving RHS named FICA. It utilizes RSUs to locally match riders by using private proximity test and private range query to protect location privacy. It achieves conditional identity privacy by using anonymous credentials. The RSUs maintain a blockchain recording ride transactions. Li et al. [12] present a privacy-preserving collaborative RHS CoRide using a consortium blockchain. It calls for a collaboration among different ride-hailing companies to tackle the problem of information isolated islands. Multiple service providers establish a consortium blockchain to support rides between riders and drivers from different platforms. Baza et al. [13] propose a decentralized ride-sharing service B-Ride based on a public blockchain. Users' locations are blurred into cloaked regions, and SC does the matching. Zero-knowledge set membership is used to prove that the driver has arrived at the current location. It requires riders to publish a ride request SC and compute distances to drivers locally. In comparison, the main privacy issue our work addresses is that riders can be linked to their anonymous requests when the drivers collude with the RHSP.

3 Problem Statement

3.1 System Architecture

The system architecture of Nereus consists of five entities: blockchain, rider, driver, RSU, RHSP, and Trusted Third Party (TTP) as shown in Fig. 2. We portray the basic (advanced) protocol in Fig. 3 (Fig. 4).

TTP. The TTP is a trusted certificate authority responsible for generating public parameters of a signature scheme, group public key, and group secret key. It also accepts users' registration requests and generates a public/private key pair and a group secret key for them.

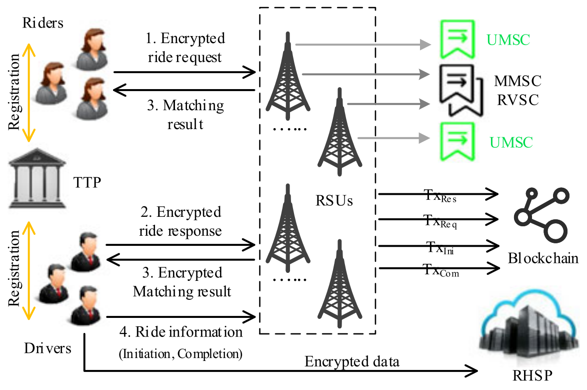


Fig. 2. System model of nereus.

RHSP. The RHSP is a ride-hailing service platform guiding users on how to use the RHS. It manages the basic information of all drivers as Didi. We reserve the role of a service provider under the public blockchain infrastructure because we do not consider full anonymity in this work and basic user information (e.g., name, cellphone, and license plate) needs to be recorded by the RHSP for potential user management and tracking, especially for vehicular digital forensics [32].

Rider. A rider requests a ride by sending an encrypted ride request to the RHSP via a local RSU. The request includes a current location and a destination. After a matching result is returned by the RSU, she will contact the driver via a cellphone number to negotiate a specific pick-up location. For clarity, we use “she” to stand for rider and use “he” to refer to driver in the following descriptions. Each rider registers a set of Ethereum accounts to participate in the blockchain network.

Driver. A driver owns a vehicle that is equipped with a tamper-proof tracking GPS device and a taximeter. He shares a ride by sending an encrypted ride response to RHSP via an RSU. The response includes a current location, a destination, and a reputation score. A ride order is formed after the driver communicates with the rider and uploads ride initiation/complete message. Riders and drivers interact with the blockchain network to send/receive messages by running atop lightweight nodes avoiding storing a complete ledger. We require that each driver uses one public key to be the network identity and link to an updatable reputation score.

RSU. An RSU is an edge node deployed by RHSP or the mobile network operator with sufficient computation and

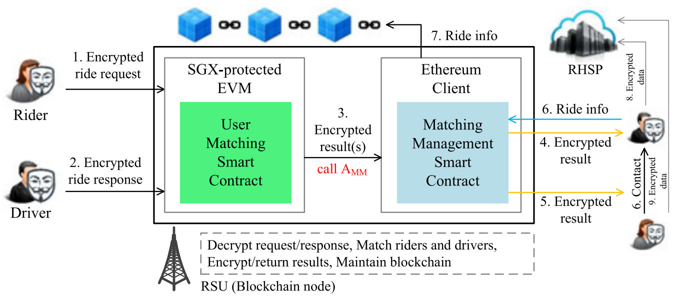


Fig. 3. Collusion-resistant user matching for RHS.

communication capabilities. RSUs match users and maintain the blockchain. They collect requests/responses from riders/drivers within their coverage. We assume that RSUs in Nereus do not share ride requests and responds with each other and only match users located in their coverage area. The area includes several cells which supports cross-cell matching. For security and privacy concerns, each RSU is equipped with SGX hardware, and it automatically matches users by executing SCs in the SGX-protected environment. Each RSU has a unique pair of keys for its enclave. After user matching, the RSU returns the result to users. We note that the proposed RHS system is built upon edge computing, all the RSUs, i.e., nodes of the blockchain, execute the same matching SC not for all transactions, but for the ones submitted to them.

Blockchain. The blockchain is a public blockchain that processes all the ride-hailing transactions. It is maintained by a group of miners who generate new blocks using a consensus mechanism. The miners include SGX-supported RSUs and normal nodes. Nereus contains three function requirements, namely: user matching, user management, and ride verifying. They are presented as SGX-protected RSUs and User Matching Smart Contract (UMSC), Matching Management Smart Contract (MMSC), and Ride Verifying Smart Contract (RVSC). We note that the integration of SGX-based SCs and Ethereum does not affect the synchrony of all the blockchain nodes. This is because even though some of the nodes do not have the SC installed inside an SGX enclave, we achieve network consensus through an Ethereum client at each RSU.

3.2 Security Model

The underlying blockchain is trusted for correctness and availability while not for user privacy [26]. The codes of SC

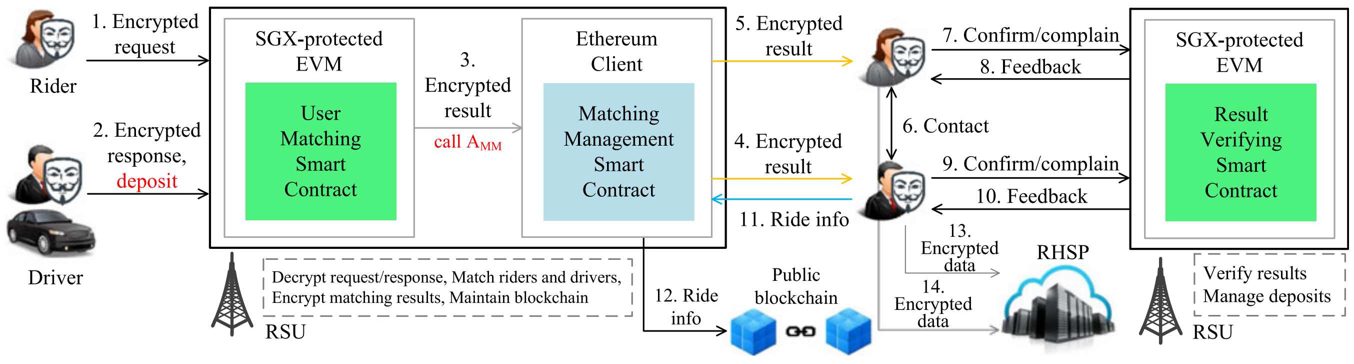


Fig. 4. Collusion resistant, verifiable, and accountable user matching for RHS.

are public. We assume that the adversary cannot physically open and manipulate SGX-enabled processor packages [29] residing in the RSUs. Thus, SGX-protected SCs run in an established and tamper-resistant manner. Specifically, threats come from untrusted RHSP, untrusted RSU, and a small number of malicious users. They are summarized as follows.

TTP. The TTP is a trusted entity that is difficult to be breached by adversaries. The adoption of TTP is a common approach in existing work [3], [4], [10], and we do not consider the existence of the TTP a privacy violation to users.

Malicious RHSP. The RHSP is curious about identities and locations, and it can collude with a driver. The RHSP cannot be trusted because: first, it may be compromised by adversaries launching cyber attacks; second, some malicious employee may steal users' information in the dataset and sell it to advertisers or black market for money; third, some malicious employee may collude with a driver to track riders and commit a crime.

Malicious RSU. An RSU faithfully matches riders with drivers and will not ignore any rider's request or driver's response. However, it is curious about users' identities and locations. Since some RSUs are deployed by the RHSP, these RSU can also engage in the collusion attack as well.

Malicious Rider. A malicious rider is curious about other identities and locations. She can send inconsistent matching information to the matched driver, i.e., current location and destination.

Malicious Driver. A malicious is curious about other identities and locations. He can collude with the RHSP to track riders' identities, refuse to accept an assigned matching result, and send inconsistent matching information to the matched rider, i.e., current location.

External Adversary. Eavesdroppers can store all the ride-hailing transactions on the blockchain aiming to acquire users' identities and locations.

Denial-of-service and physical attacks, e.g., power analysis, are out of the scope of this paper [31]. Moreover, social engineering, e.g., taking a photo of the rider by a hidden camera, is not considered.

3.3 Design Objectives

Our design objectives are three-fold: privacy, security, and efficiency. Informally, the privacy and security requirements can be stated as follows. After the execution of an Nereus protocol, any participating user u_i cannot learn (1) the identity id_j of any other user u_j, $u_j \neq u_i$ nor (2) the current location cl_j, destination dn of any other user u_j, $u_j \neq u_i$. We exclude one case that after picking up the matched rider, a driver knows the cl_j and dn_j of the matched rider. This is inevitable due to the service requirement. Similarly, any RSU and the RHSP cannot learn (3) the identity id_j of any other user u_j, $u_j \neq u_i$ nor (4) the current location cl_j, destination dn of any other user u_j, $u_j \neq u_i$. For security, we explicitly give the adversaries the ability to collude.

We construct corresponding cryptographic games in which the adversary interacts with the algorithm and we define the adversary's winning advantage. We say the algorithm is privacy-preserving and secure if the adversary's advantage in each game is negligible. Since there are different types of adversaries and two protocols, we will detail the game in formal proofs in Section 5. Formally, these privacy and requirements can be defined below.

Privacy. The privacy requirement is twofold. (1) Identity privacy: the real identity of rider is protected from adversaries during the ride. (2) Location privacy: no adversaries except for the matched driver/rider can acquire the exact locations (current location and destination) of the rider/driver.

First, we compute the advantage of an adversary $\mathcal{A}$ in correctly guessing the identity id_i of any user u_i. We call this advantage the identity indistinguishability advantage and denote it by $\mathsf{Adv}_{\mathcal{A}}^{\mathrm{Ide}}(\Pi)$. We define $\mathsf{Adv}_{\mathcal{A}}^{\mathrm{Ide}}(\Pi)$ by using a challenge-response game. Let $\mathcal{U} = \{u_1, u_2, \ldots, u_n\}$ be the set of all users, including the Probabilistic Polynomial-Time (PPT) adversary $\mathcal{A}$, the challenger $\mathcal{C}$, and Π the proposed basic (advanced) Nereus protocol. The *identity indistinguishability game* is defined as follows:

1. Challenger setup: $\mathcal{C}$ collects the identities id_i, $i \in \{1, \ldots, n\}$.
2. Algorithm execution: $\mathcal{C}$ executes Π with all users $\mathcal{U}$ and matches riders with drivers. It sends a matching result to each user u_i.
3. Challenge: $\mathcal{C}$ chooses a random $k \in \{1, 2, \ldots, n\}$ and sends id_i to $\mathcal{A}$.
4. Guess: $\mathcal{A}$ selects a value $k' \in \{1, 2, \ldots, n\}$ and sends it to $\mathcal{C}$. $\mathcal{A}$ wins the game if $k' = k$.

Finally, $\mathsf{Adv}_{\mathcal{A}}^{\mathrm{Ide}}$ is the advantage of $\mathcal{A}$ in winning the identity indistinguishability game: $\mathsf{Adv}_{\mathcal{A}}^{\mathrm{Ide}}(\Pi) = \Pr[\mathcal{A}\ wins] - \frac{1}{n} = \Pr[k' = k] - \frac{1}{n}$, where $\mathcal{A} \notin \mathcal{U}$, i.e., $\mathcal{A}$ is the RHSP or an RSU, and $\Pr[k' = k]$ is the probability that $\mathcal{A}$ correctly guesses k chosen by $\mathcal{C}$. When $\mathcal{A} \in \mathcal{U}$, $\mathsf{Adv}_{\mathcal{A}}^{\mathrm{Ide}}(\Pi) = \Pr[k' = k] - \frac{1}{n-1}$. Here, we do not consider it a successful attack when $u_k = \mathcal{A}$.

Second, we compute the advantage of $\mathcal{A}$ in correctly guessing the current location cl_i and destination dn_i of any user u_i. We call this advantage the location indistinguishability advantage and denote it by $\mathsf{Adv}_{\mathcal{A}}^{\mathrm{Loc}}(\Pi)$. The *location indistinguishability game* is defined as follows:

1. Challenger setup: $\mathcal{C}$ collects the current locations cl_i and destinations dn_i, $i, j \in \{1, ..., n\}$.
2. Algorithm execution: $\mathcal{C}$ executes Π with all users $\mathcal{U}$ and matches riders with drivers. It sends a matching result to each user u_i.
3. Challenge: $\mathcal{C}$ chooses a random $k \in \{1, 2, \ldots, n\}$ and sends cl_i and dn_i to $\mathcal{A}$.
4. Guess: $\mathcal{A}$ selects a value $k' \in \{1, 2, \ldots, n\}$ and sends it to $\mathcal{C}$. $\mathcal{A}$ wins the game if $k' = k$.

Finally, $\mathsf{Adv}_{\mathcal{A}}^{\mathrm{Loc}}$ is the advantage of $\mathcal{A}$ in winning the location indistinguishability game: $\mathsf{Adv}_{\mathcal{A}}^{\mathrm{Loc}}(\Pi) = \Pr[\mathcal{A}\ wins] - \frac{1}{m^2} = \Pr[k' = k] - \frac{1}{m^2}$, where $\Pr[k' = k]$ is the probability that $\mathcal{A}$ correctly guesses k chosen by $\mathcal{C}$.

Security Level I. Nereus should prevent the collusion attack between the RHSP and a malicious driver during user matching, i.e., the colluding parties cannot link the request of the matched rider whom the malicious driver picks up.

We enhance the adversary's power by considering the ability to corrupt [33]. Specifically, we model this ability as a corrupt query $\mathsf{Corrupt}()$. When $\mathcal{A}$ issues a query $u_{\hat{c}}$ to $\mathsf{Corrupt}()$, $\mathcal{C}$ returns all the possessed information $\mathcal{I}_{\hat{c}}$ to $\mathcal{A}$, where $u_{\hat{c}}$ is the colluding driver.

We compute the advantage of $\mathcal{A}$, i.e., the RHSP or an RSU, colluding with a driver $d_{\hat{c}}$ in correctly guessing the identity id_i of any user u_i. We call this advantage the collusion advantage and denote it by $\mathsf{Adv}_{\mathcal{A}}^{\text{Col}}(\Pi)$. The *collusion resistance game* is defined as follows:

1. Challenger setup: $\mathcal{C}$ collects the identities id_i, $i \in \{1, ..., n\}$.
2. Algorithm execution: $\mathcal{C}$ executes Π with all users $\mathcal{U}$ and matches riders with drivers. It sends a matching result to each user u_i.
3. Challenge: $\mathcal{C}$ chooses a random $k \in \{1, 2, \ldots, n'\}$ and sends id_j to $\mathcal{A}$.
4. Corrupt: $\mathcal{A}$ makes queries to $\mathsf{Corrupt}()$ and receives $\mathcal{I}$.
5. Guess: $\mathcal{A}$ selects a value $k' \in \{1, 2, \ldots, n'\}$ and sends it to $\mathcal{C}$. $\mathcal{A}$ wins the game if $k' = k$.

Finally, $\mathsf{Adv}_{\mathcal{A}}^{\text{Col}}$ is the advantage of $\mathcal{A}$ in winning the collusion resistance game: $\mathsf{Adv}_{\mathcal{A}}^{\text{Col}}(\Pi) = \Pr[\mathcal{A}\ wins] - \frac{1}{n'} = \Pr[k' = k] - \frac{1}{n'}$, where $\mathcal{A}$ is a driver, n' is the number of riders, and $\Pr[k' = k]$ is the probability that $\mathcal{A}$ correctly guesses k chosen by $\mathcal{C}$.

Security Level II. Based on security level I, Nereus should guarantee the matching verifiability and accountability, namely users are able to verify the returned matching result and runaway users are punished. Specifically, verifiability has two requirements, i.e., the returned user is a valid match and the user is the best match. A user gets compensated if the matched user runs away.

Efficiency. Nereus should be efficient, i.e., the computational costs and communication overhead should be acceptable.

4 The Proposed Nereus

To achieve the two defined security levels, we present a basic protocol and an advanced protocol. The basic protocol consists of five phases: initialization and registration, ride request, ride response, ride matching, and ride initiation and complete. The advanced protocol includes an additional ride verifying phase. Table 1 shows the notations frequently used in Nereus.

TABLE 1
Key Notations of Nereus

Notation	Definition
RHS, RHSP	Ride-hailing service, RHS provider
PSC, PB	Private smart contract, public blockchain
PPT, $\mathcal{A}$, $\mathcal{C}$	Probabilistic polynomial-time, adversary, challenger
$u_{\hat{c}}$, $\mathcal{I}$	Colluding driver, all possessed information
$\mathcal{GC}$, m, c	Set of geographic cells, number of cells, cell
p, g	Big prime, group generator
$\mathcal{BF}$, l, $\mathcal{H}$	Bloom Filter, length of $\mathcal{BF}$, set of hash functions
h, k	Hash function, number of h
$\mathrm{A_{MM}}$, $\mathrm{A_{UM}}$	Address of MMSC, address of UMSC
rs, no	Reputation score, number of ride orders
(pk, pi), gsk	Public/private key pair, group secret key
cl, dn, c	Current location, destination, cell index
cp, rn, ts	Cellphone, random number, timestamp
rr_R, er_R	Ride request, encrypted ride request
cl, pd	Current location, potential destination
rr_D, er_D	Ride response, encrypted ride response
ro, mr	Rider order, encrypted matching result
rf, ed, $data$	Ride fare, encrypted user data, user data
$\mathcal{IBF}$, t	Indistinguishable Bloom filter, number of twins
$H_1, H_2, \ldots, H_z$	Pseudo-random hash functions
$\mathcal{PF}$, pr, $\mathcal{MS}$	Prefix family, prefix, minimum sets of prefixes
d, $comm$	Default deposit value, commitment
Mp, N	Merkle proof, number of leaf nodes in Merkle tree
σ_{Req}, σ_{Res}; dis	Group signature; euclidean distance

4.1 The Basic Protocol

4.1.1 Initialization and Registration

The ride hailing area is divided by the RHSP into a set of m geographic cells $\mathcal{GC} = \{c_1, c_2, \ldots, c_m\}$. Cells are defined according to roads and blocks. For example, a cell could be a grid of 100×100 meters^2. Cells are arranged in different levels such that a cell of a higher level contains the cells of a lower level. For anonymous authentication, the TTP generates public parameters and keys according to [34], including a big prime p and a group generator g. Since this part is not the core of our scheme, we omit the detailed descriptions. A CCA2-secure public key cryptosystem $\Sigma = (\mathsf{S}, \mathsf{E}, \mathsf{D})$ is chosen with algorithms Setup, Encryption, and Decryption. A length l of bloom filter $\mathcal{BF}$ is set and k secure hash functions are chosen as $\mathcal{H} = \{h_1, h_2, \ldots, h_k\}$ where $h_i : \{0,1\}^* \rightarrow \{0,1\}^l$.

The public blockchain is denoted as PB. Each RSU shares the same MMSC but has its own UMSC. The MMSC is deployed on PB with a unique address $\mathrm{A_{MM}}$ the same way it is done in Ethereum. But the deployment of each UMSC is different from the MMSC because it is SGX-protected. The UMSC also has a unique address $\mathrm{A_{UM}}$. Two pools $rs\{\}$ and $no\{\}$ are initialized to store drivers' reputation score rs and number of ride orders no. The two public pools are stored at the MMSC on PB. They are two global variables that can be accessed by all blockchain entities. Each rider (driver) registers with the TTP to obtain a set of public/private key pairs $\{(pk_R, pi_R)\}$ ($\{(pk_D, pi_D)\}$) and a group secret key gsk_R (gsk_D). The gsk includes an element g^u where u is randomly chosen by the TTP for each user and is considered as the user's digital identity [34]. Next, each RSU U registers with the TTP to obtain a public/private key pair (pk_U, pi_U). $(pk_{\text{UM}}, pi_{\text{UM}})$ is generated for the UMSC and stored at U by SGX Seal method. The identity of a SC is a hexadecimal address. It is used when calling a function in a SC.

4.1.2 Ride Request

A rider R_i standing at a current location cl_i within the area of RSU U wishes to leave for a destination dn_i. She transforms cl_i into a cell index c_{R_i} that covers her cl_i. The c_{R_i} is attached with a mark 0 to stand for current location while 1 for destination later. (This is to prevent matching $A \rightarrow B$ with $B \rightarrow A$.) Then she hashes it to k locations $\mathcal{BF}_{R_i}[h_1(c_{R_i})], \mathcal{BF}_{R_i}[h_2(c_{R_i})], \ldots, \mathcal{BF}_{R_i}[h_k(c_{R_i})]$ in a Bloom filter $\mathcal{BF}_{R_i}$. Similarly, she inserts dn_i into $\mathcal{BF}_{R_i}$. Next, R_i generates a public/private key pair (pk_{R_i}, pi_{R_i}). For communication with a driver later, R_i will use her cellphone cp_{R_i}. She receives the pk_{UM} in the broadcast of U, generates a random number rn_i, and encrypts her ride request $rr_{R_i} = (\mathcal{BF}_{R_i} || pk_{R_i} || cp_{R_i})$ to form an encrypted ride request $er_{R_i} = \mathsf{E}(pk_{\text{UM}}, pk_{R_i} || rr_i || rn_i)$. Finally, R_i sends er_{R_i} with a group signature $\sigma_{R_i}^{\text{Req}}$ signed by gsk_{R_i} to PB in the form of a ride request transaction:

$$Tx_{R_i}^{\text{Req}} = (\text{``Request''}, U, er_{R_i}, ts_{R_i}^{\text{Req}}, \sigma_{R_i}^{\text{Req}}), \tag{1}$$

where ts is a timestamp, i.e., the time at which the rider sends the request transaction. The pk_{R_i} is transmitted to U's enclave which will use SGX Seal method to store (pk_{R_i}). pk_{R_i} will be used to encrypt the user matching result.

4.1.3 Ride Response

A driver D_j cruising around a current location cl_j is ready to take a rider to a potential destination pd_j. He transforms cl_j and pd_j into cell indexes $c_{D_j}^1$ and $c_{D_j}^2$, respectively. Next, D_j generates a public/private key pair (pk_{D_j}, pi_{D_j}). He generates a random number rn_j, and encrypts $rr_{D_j} = (c_{D_j}^1 || c_{D_j}^2 || pk_{D_j} || cp_{D_j})$ to form an encrypted ride response $er_{D_j} = \mathsf{E}(pk_{\text{UM}}, pk_{D_j} || rr_{D_j} || rn_j))$. Finally, D_j sends a ride response transaction to PB including er^{D_j} and a group signature $\sigma_{D_j}^{\text{Res}}$ signed by gsk_{D_j}:

$$Tx_{D_j}^{\text{Res}} = (\text{``Response''}, U, er_{D_j}, ts_{D_j}^{\text{Res}}, \sigma_{D_j}^{\text{Res}}). \tag{2}$$

4.1.4 Ride Matching

After receiving the ciphertext er_{R_i}, U verifies $\sigma_{R_i}^{\text{Req}}$ and then decrypts er_{R_i} with pi_{UM} to read rr_{R_i} into the enclave. Next, the enclave decrypts and stores the request by calling the function Request in Algorithm 1. Similarly, the enclave calls the function Response and reads rr_{D_j}. The enclave carries out user matching by calling the function Match. The function proceeds by hashing each candidate driver's cell indexes $c_{D_j}^1$, $c_{D_j}^2$ into a rider's k locations in $\mathcal{BF}_{R_i}$. If all the locations are occupied when checking one prefix, the current driver is a matching candidate. By doing so, we have separated the matching process from the RSU by utilizing the enclave. Recalling the Alice and Bob example in Section 1, the colluding Bob and the RHSP (RSU) will not learn any information on the link between Alice's ride request and her matching result.

The matching result output by UMSC is threefold. (1) If only one driver meets the rider's conditions, the enclave generates a rider order ro_{ij}, i.e., an integer, and returns an encrypted matching result (outside of the enclave by a local call to A_{MM}) $mr_U^{R_i} = (\text{``Match''}, \mathsf{E}(pk_{R_i}, ro_{ij} || pk_{D_j} || cp_{D_j}))$ $(mr_U^{D_j})$ and a signature $\sigma_{\text{UM}}^{R_i}$ $(\sigma_{\text{UM}}^{D_j})$ signed by pi_{UM} to MMSC. MMSC forwards them to R_i (D_j) after verifying their validity. (2) If more than one driver satisfies the matching conditions, the enclave returns a set of encrypted matching results $mr_U^{R_i} = (\text{``Multiple''}, \{pk_{D_j}, \mathsf{E}(pk_{R_i} || pk_{D_j} || cp_{D_j})\})$ and a signature to the MMSC. The MMSC will select a driver with the highest reputation score rs_j and returns one result to R_i. It is achieved by calling the function Manage in Algorithm 2. In the two cases above, MMSC will also send an encrypted feedback to the UMSC to delete the associated driver by calling the function Delete in Algorithm 1. (3) If no qualified driver is found, UMSC will send a waiting message "Wait" and a signature via the MMSC to R_i and continue to wait for a qualified driver. R_i can choose to exit or keep waiting.

Algorithm 1. Pseudocode for UMSC

```
1: create Req; //Ride request
2: mapping (bytes32 => uint256) Res; //Map of ride response
3: mapping (bytes32 => uint256) MR; /Map of matching result
4: function Request($rr_R$)
5:     Req = $rr_R$;
6:     if (no deposit) call BasicMatch;
7:     else call AdvancedMatch;
8: function Response($rr_D$)
9:     Res{} ← $rr_D$;
10: function BasicMatch(Req, Res{}, MR{})
11:     while not at end of Res{}
12:         if ($\mathcal{BF}[c_D^1] \neq 1 | \mathcal{BF}[c_D^2] \neq 1$) continue;
13:         else MR{} ← $rr_D$;
14:     return MR{}; /*Ready to encrypt MR{} with $pk_D$ before leaving the enclave*/
15: function Delete($rr_D$)
16:     Delete ($rr_D$, Res{});
17: function AdvancedMatch(Req, Res{})
18:     while not at end of Res{}
19:         if ($\mathcal{IBF}_R[H_o(pr)][H'(H_{z+1}(H_o(pr)) \oplus rn_R)] = 1 \&\& \mathcal{IBF}_R[H_o(pr)][1 - H'(H_{z+1}(H_o(pr)) \oplus rn_R)] = 0$)
20:             MR{} ← $rr_D$;
21:         else continue;
22:     return MR{};
```

4.1.5 Ride Initiation and Complete

After receiving $mr_U^{R_i}$ from the MMSC, R_i decrypts it with pi_{R_i} and contacts D_j through cp_{D_j} to negotiate a specific pick-up location. ro_{ij} is formed by D_j's uploading a ride initiation transaction to PB:

$$Tx_{D_j}^{\text{Ini}} = (\text{``Initiation''}, ro_{ij}, pk_{D_j}, pk_{R_i}, ts_{D_j}^{\text{Ini}}, \sigma_{R_i}^{\text{Ini}}, \sigma_{D_j}^{\text{Ini}}). \tag{3}$$

Meanwhile, rr_{R_i} and rr_{D_j} are deleted from the enclave.

After arriving at her destination, the ride fare rf_{ij} is calculated by the taximeter and R_i pays rf_{ij} either by a transaction or mobile payment. D_j submits a ride complete transaction to the MMSC:

$$\begin{aligned} Tx_{D_j}^{\text{Com}} = (\text{``Complete''}, ro_{ij}, pk_{D_j}, pk_{R_i}, rt_{R_i}, \\ ts_{D_j}^{\text{Com}}, \sigma_{R_i}^{\text{Com}}, \sigma_{D_j}^{\text{Com}}). \end{aligned} \tag{4}$$

where rt_{R_i} is a rating. The reputation score rs_j of D_j is updated only if a ride has been successfully completed. It is achieved by $Tx_{D_j}^{\text{Com}}$'s calling the function Complete in Algorithm 2.

Finally, D_j and R_i send their encrypted user data ed_{D_j} and ed_{R_i} to the RHSP for storage where

$$\begin{aligned} ed_{D_j} &= \mathsf{E}(pk_{D_j}, data_{D_j}), ed_{R_i} = \mathsf{E}(pk_{R_i}, data_{R_i}), \\ data_{D_j} &= cl_j || dn_j || pk_{D_j} || cp_{D_j} || \sigma_{R_i}^2 || \sigma_{R_i}^3 || ts_{D_j}^{\text{Ini}} || ts_{D_j}^{\text{Com}}, \\ data_{R_i} &= cl_i || dn_i || pk_{R_i} || cp_{R_i} || rt_{R_i} || ts_{D_j}^{\text{Ini}} || ts_{D_j}^{\text{Com}}. \end{aligned}$$

This step is designed for self-query and vehicular digital forensics. First, users can query their previous ride data from the RHSP. Second, when a dispute or a road accident happens, an investigator is looking for pertinent evidence.

The rider or driver who is an involved party can pass the temporary private key to the investigator to retrieve and decrypt the stored data on RHSP.

Algorithm 2. Pseudocode for MMSC

```
1: mapping (bytes32 => uint256) no; //Map of number of orders
2: mapping (bytes32 => uint256) rs; //Map of reputation scores
3: create Add{}, Dep{}, Com{}; //Sets of addresses, deposits, commitments
4: function Manage("Multiple" , $\{pk_D\}, mr_U^R)\}$)
5:     find driver with the highest $rs$;
6:     return the found $\mathsf{E}(pk_R, pk_D||cp_D)$;
7: function Complete ("Complete", $pk_D, pk_R, rt_R, ts_D, \sigma_R, \sigma_D$)
8:     $no[pk_D] \leftarrow no[pk_D] + 1$; //Update no
9:     $rs[pk_D] \leftarrow \frac{rs[pk_D]*no[pk_D]+rt_R}{no[pk_D]}$; //Update rs
10: function Deposit("Deposit", $pk, com, d, ts, \sigma$)
11:     $Add\{\} \leftarrow pk$; $Dep\{\} \leftarrow d$; $Com\{\} \leftarrow com$;
12: function Transfer("Transfer", $pk_1, pk_2, d$)
13:     Dep[$pk_1$] $\leftarrow$ Dep[$pk_1$] + $d$; Delete($pk_2$, Add, Dep, Com);
14: function Complain("Complain", $pk_1, pk_2$)
15:     publish ("Complain", $pk_1, pk_2$);
16:     if (block.timestamp < expiration && receive valid response)
17:         call Transfer("Transfer", $pk_2, pk_1, d$);
18:     else call Transfer("Transfer", $pk_1, pk_2, d$);
```

4.2 The Advanced Protocol

In the advanced protocol, we guarantee the verifiability and accountability by designing an RVSC integrating privacy-preserving range query, Merkle proofs, deposit commitments, and "Proof-of-Runaway". To favor readability, we omit the detailed descriptions of the same steps in the basic protocol, but present the RVSC in detail.

4.2.1 Initialization and Registration

TTP chooses the number of twins t for an indistinguishable Bloom filter $\mathcal{IBF}$, generates $z+1$ secret keys $K_1, K_2, \ldots, K_{z+1}$, constructs z pseudo-random hash functions $H_1, H_2, \ldots, H_z$ where $H_i(.) = \text{HMAC}_{K_i}(.)\%t$ for $1 \leq i \leq z$, constructs two pseudo-random hash functions $H_{z+1}(.) = \text{HMAC}_{K_{z+1}}(.)$, $H'(.) = \text{SHA256}(.)\%2$, where HMAC is a keyed-hash message authentication code [35]. Each user registers with the TTP to receive the secret keys, pseudo-random hash functions, a group secret key, and a set of key pairs. Each RSU holds an additional RVSC with an address A_{RV}. The deployment of RVSC is similar to the UMSC, which is also SGX-protected. A public/private key pair $(pk^{\text{RV}}, pi^{\text{RV}})$ is generated for the RVSC and stored at U by SGX Seal method.

4.2.2 Ride Request

For the current cell index c_{R_i}, R_i converts it into a prefix family $\mathcal{PF} = \{pr_v\}$ [35] and hashes each prefix pr into z twins $\mathcal{IBF}_{R_i}[H_1(pr)], \ldots, \mathcal{IBF}_{R_i}[H_z(pr)]$. For $\mathcal{IBF}_{R_i}[H_o(pr)]$ $(1 \leq o \leq z)$, R_i uses H' to determine the chosen cell location $H'(H_{z+1}(H_o(pr)) \oplus rn_{R_i})$ where rn_i is a random number. Then, R_i sets $\mathcal{IBF}_{R_i}[H_o(pr)][H'(H_{z+1}(H_o(pr)) \oplus rn_{R_i})] = 1$ and $\mathcal{IBF}_{R_i}[H_o(pr)][1 - H'(H_{z+1}(H_o\,(pr)) \oplus rn_{R_i})] = 0$. Similarly, R_i inserts the prefix family of destination cell index into $\mathcal{IBF}_{R_i}$. The ride request is $rr_{R_i} = (\mathcal{IBF}_{R_i}, rn_{R_i}, pk_{R_i}, cp_{R_i}, in_{R_i})$, where in_{R_i} is the rider's features. R_i encrypts rr_{R_i} to obtain $er_{R_i} = \mathsf{E}(pk_{\text{UM}}, rr_{R_i}||rn'_i)$. When requesting a ride, R_i is required to deposit some money [36] d (a default value specified by the system) to a UMSC deployed on a nearby RSU and MMSC. The d is regarded as a commitment to her ride order $com_{R_i} = H_1(d||rr_{R_i})$. It is achieved by sending a deposit transaction calling function Deposit in Algorithm 2 and enclave's calling the function Request in Algorithm 1:

$$Tx_{R_i}^{\text{Dep}} = (\text{“Deposit”}, U, pk_{R_i}, d, com_{R_i}, er_{R_i}, ts_{R_i}^{\text{Dep}}, \sigma_{R_i}^{\text{Dep}}). \quad (5)$$

After the MMSC stores the rider deposit d, the rider can claim it back if there are no available drivers or the matched driver misbehaves. The deposit will be transferred the driver if the rider arrives at her destination.

4.2.3 Ride Response

For the current cell index $c_{D_j}^1$ and the destination cell index $c_{D_j}^2$, D_j can choose to expand them to two new cell indexes of higher level to increase the probability of taking a ride order. D_j converts them into two minimum sets of prefixes $\mathcal{MS}$ [35] and computes a ride response $rr_{D_j} = \{(H_o(pr), H_{z+1}(H_o(pr)))\}_{o=1}^{z}, pk_{D_j}, cp_{D_j}, in_{D_j})$ for each prefix in the two sets. Next, D_j computes a commitment $com_{D_j} = H_1(d||rr_{D_j})$ and sends a deposit transaction:

$$Tx_{D_j}^{\text{Dep}} = (\text{“Deposit”}, U, pk_{D_j}, d, com_{D_j}, er_{D_j}, ts_{D_j}^{\text{Dep}}, \sigma_{D_j}^{\text{Dep}}). \quad (6)$$

After the MMSC stores the driver deposit d, the driver can claim it back if there are no available riders or the matched rider behaves.

4.2.4 Ride Matching

Similar to the basic protocol, the RSU U reads rr_{R_i} and rr_{D_j} into the enclave to carry out user matching. It proceeds by hashing prefixes into t twins in $\mathcal{IBF}_R$, i.e., checking whether $\mathcal{IBF}_R[H_o(pr)][H(H_{z+1}(H_o(pr)) \oplus rn_R)] = 1$ and $\mathcal{IBF}_R[H_o(pr)][1 - H(H_{z+1}(H_o(pr)) \oplus rn_R)] = 0$ for $1 \leq o \leq z$. If all the twins are occupied when checking one prefix, then the current driver is a matching candidate. The matching results including a temporary rider order ro_{ij} and a signature are sent to the two matched users. Additionally, we require that only one response is selected for the rider. UMSC stores the matched request and response to RVSC for result verification later.

4.2.5 Ride Verifying

After receiving the matching result from the UMSC, R_i contacts D_j to negotiate a current location cl_{ij}. To verify the matching result, R_i sends a ciphertext of $ro_{ij}||rr_{R_i}||cl_{ij}$ to the RSU. D_j sends a ciphertext of $ro_{ij}||rr_{D_j}||cl_{ij}$ to the RSU. The enclave within the RSU decrypts the two ciphertexts and calls the function Verify in Algorithm 3. It proceeds by first computing two new hash values of the response and the request, and then comparing them with the commitments stored in the MMSC.

If D_j's input does not produce his commitment, the verification fails. The RVSC sends a feedback "False input from driver." to R_i, calls the function Transfer

(pk_{R_i}, pk_{D_j}, d) in Algorithm 3, and sends encrypted rr^{D_j} to the UMSC which deletes rr_{D_j} by calling the function Delete(rr_D) to remove rr_{D_j} from Res{}. If R_i provides a false input, the RVSC sends a feedback "False input from rider." to D_j and calls Transfer(pk_{D_j}, pk_{R_i}, d). Otherwise, the RVSC continues to check if rr_{D_j} matches rr^{R_i}. If they do not match, the RVSC sends a feedback "Not qualified." to the rider or driver.

If correctly comparing with two commitments, the verification succeeds. The RVSC computes the $log_2 N + 1$ (where N is the number of leaf nodes) Merkle proofs Mp_R^{ij}/Mp_D^{ij} (including the root value) on current set $\{IBF_R, rn_R\}/\{rr_D(1)\}$ and sends them to R_i/D_j. Here, $rr_{D_j}(1)$ is the first item of rr_{D_j}, i.e., $\{(H_o(pr), H_{z+1}(H_o(pr)))\}_{o=1}^{z}$. The order of the leaf nodes in a Merkle tree is based on their timestamps.

4.2.6 Ride Initiation and Complete

After verifying the received Merkle proofs, the tamper-proof GPS device of D_j is triggered to sends a current location cl_j to the RSU. By calling the function `CalculateDis` in Algorithm 3, the enclave in RSU can calculate the euclidean distance $dis = d_E(cl_j, cl_{ij})$ between cl_j and cl_{ij}. If D_j abandons R_i before the ride begins, R_i submits a complaint message to the MMSC. If not receiving a location from D_j or the calculated dis is bigger than a threshold (e.g., 50 meters), the associated two deposits will be transferred to R_i and the temporary rider order ro_{ij} is terminated. The abnormal distance is considered as the PoR. Otherwise, D_j uploads ride initiation transaction $Tx_{\text{Ini}}^{D_j}$ including $(IBF_{R_i}, rn_{R_i}, rr_{D_j}(1),\ Mp_R^{ij}, Mp_D^{ij})$ to the MMSC. In this way, anyone in the blockchain network can verify the matching result by comparing the Merkle proofs with the computation result on the initiation transaction.

Algorithm 3. Pseudocode for RVSC

```
1: create VReq{}, VRes{}; //Two sets of ride requests and responses
2: create MProofR, MProofD; //Proofs returned to rider and driver
3: function Verify($rr_R$, $rr_D$)
4:    compute commitments $Com_R$;
5:    compute commitments $Com_D$;
6:    if ($Com_R \neq com[pk_R]$)
7:       return "False input from rider";
8:    if ($Com_D \neq com[pk_D]$)
9:       return "False input from driver";
10:   VReq{} ← $IBF^R$; VReq{} ← $rn_R$; VRes{} ← first item of $rr_D$;
11:   compute MProofR/MProofD from current VReq{}/VRes{};
12:   return (MProofR, MProofD);
13: function CalculateDis($cl$, $cl$)
14:   calculate $dis = d_E(cl, cl)$;
```

After arriving at R_i's destination, D_j submits a ride complete transaction $Tx_{D_j}^{\text{Com}}$ to the MMSC to trigger the deposit transfer Transfer(pk_{D_j}, pk_{R_i}, d) and update D_j's reputation score. R_i pays rf_{ij} deducting the transferred deposit to D_j either by a transaction or mobile payment. Finally, D_j and R_i send ed_{D_j} and ed_{R_i} to the RHSP for storage.

5 Privacy and Security Analysis

In this section, we formally prove the privacy and security of Nereus with respect to the adversary model and design objectives. In both protocols, we utilize short group signatures to provide anonymous authentication, so we prove the identity privacy of the two protocols in one theorem. For location privacy and security level I, we prove with two theorems.

5.1 Privacy

Theorem 1. *Our basic (advanced) Nereus scheme is* $\mathsf{Adv}_{\mathcal{A}}^{\text{Ide}}(\Pi)$ *-identity indistinguishable against PPT adversaries under the Discrete Logarithm (DL) assumption, where* $\mathsf{Adv}_{\mathcal{A}}^{\text{Ide}}(\Pi) \leq \frac{1}{2^{|g|}}$.

Proof: We have four types of adversaries, driver, rider, RSU, and RHSP. Hence, we prove according to the four types. We note that rider and driver is interchangeable, so their proofs are similar. RSU and RHSP are not one of the users, so they are treated as an observer.

(1) Driver as $\mathcal{A}$. There are n cases for Π: i drivers and $n-i$ riders, $1 \leq i \leq n$. When $i = 1$, the matching result can be a successful match with one rider or an unsuccessful match with all $n-1$ riders. In both scenarios where $u_k \neq \mathcal{A}$, $\mathcal{A}$ can correctly guess k by (1) correctly guessing the value of u of g^u belonging to the group secret key gsk [34] or (2) randomly guessing k. In both scenarios where $u_k = \mathcal{A}$, $\mathcal{A}$ correctly guesses k with the probability of 1. However, we do not consider this scenario as a successful attack since the chosen challenge k refers to the adversary itself. As $\mathsf{Adv}_{\mathcal{A}}^{\text{Ide}}(\Pi) = \Pr[k' = k] - \frac{1}{n-1}$, we have $\mathsf{Adv}_{\mathcal{A}}^{\text{Ide}}(\Pi) = \Pr[k' = k] - \frac{1}{n-1} = \frac{n-1}{n} * \frac{1}{2^{|g|}} + \frac{1}{n-1} - \frac{1}{n-1} = \frac{n-1}{n} * \frac{1}{2^{|g|}} < \frac{1}{2^{|g|}}$, which is negligible under the DL problem.

(2) Rider as $\mathcal{A}$. There are n cases for Π: i riders and $n-i$ drivers, $1 \leq i \leq n$. This type is similar to the first type, which means $\mathsf{Adv}_{\mathcal{A}}^{\text{Ide}}(\Pi) < \frac{1}{2^{|g|}}$.

(3) RSU as $\mathcal{A}$. A malicious RSU is outside of the user set and its target is among the n users. Although it handles the matching for the n users, the plaintexts are in the enclave and hidden from the RSU. We have $\mathsf{Adv}_{\mathcal{A}}^{\text{Ide}}(\Pi) = \Pr[k' = k] - \frac{1}{n} = \frac{1}{2^{|g|}} + \frac{1}{n} - \frac{1}{n} = \frac{1}{2^{|g|}}$, which is negligible.

(4) RHSP as $\mathcal{A}$. The malicious RSU is also outside of the user set. Similar to the RSU type, we have $\mathsf{Adv}_{\mathcal{A}}^{\text{Ide}}(\Pi) = \frac{1}{2^{|g|}}$.

To summarize, the basic (advanced) Nereus scheme is $\mathsf{Adv}_{\mathcal{A}}^{\text{Ide}}(\Pi)$-identity indistinguishable against PPT adversaries.

Theorem 2. *Our basic Nereus scheme* Π_1 *is* $\mathsf{Adv}_{\mathcal{A}}^{\text{Loc}}(\Pi_1)$-*-location indistinguishable against PPT adversaries, where* $\mathsf{Adv}_{\mathcal{A}}^{\text{Ide}}(\Pi_1) \leq \frac{1}{2^{|pi|}}$.

Proof: We also have two types of locations: rider's current location cl and destination dn inserted into a Bloom filter $\mathcal{BF}$ and driver's cell indexes (c_1^D, c_2^D). They are all encrypted under the enclave's public key. Since the malicious driver and malicious rider could be seen as the same type of adversary, and so does the RSU and RHSP, we proceed with the two types of adversaries, driver and rider, RSU and RHSP.

(1) Driver and rider as $\mathcal{A}$ and $\mathcal{A}$. There are n cases for Π: i drivers and $n-i$ riders, $1 \leq i \leq n$. $\mathcal{A}$ tries to recover the plaintext of er_R or er_D, which is rr_R or rr_D. When $u_k \neq \mathcal{A}$, the adversary can correctly guess k by (1) correctly guessing the value of private key pi^{UM} of the enclave and the cell indexes of current location and destination or (2) randomly guessing the cell indexes of current location and destination which equals to $\frac{1}{m^2}$. (m is the number of cells.) As $\mathsf{Adv}_{\mathcal{A}}^{\text{Loc}}(\Pi) = \Pr[k' = k] - \frac{1}{m^2}$, we have $\mathsf{Adv}_{\mathcal{A}}^{\text{Loc}}(\Pi_1) = \Pr[k' = k] - \frac{1}{m^2} =$

$\frac{n-1}{n} * \frac{1}{2^{|pi|}} + \frac{1}{m^2} - \frac{1}{m^2} = \frac{n-1}{n} * \frac{1}{2^{|pi|}} < \frac{1}{2^{|pi|}}$, which is negligible under the security of the public key cryptosystem.

(2) RSU and RHSP as $\mathcal{A}$. A malicious RSU or RHSP aims to recover rr_R or rr_D. We have $\mathsf{Adv}_{\mathcal{A}}^{\mathrm{Loc}}(\Pi_1) = \Pr[k' = k] - \frac{1}{m^2} = \frac{1}{2^{|pi|}} + \frac{1}{m^2} - \frac{1}{m^2} = \frac{1}{2^{|pi|}}$, which is negligible.

To summarize, the basic Nereus scheme Π_1 is $\mathsf{Adv}_{\mathcal{A}}^{\mathrm{Loc}}(\Pi_1)$ -location indistinguishable against PPT adversaries.

Theorem 3. *Our advanced Nereus scheme Π_2 is $\mathsf{Adv}_{\mathcal{A}}^{\mathrm{Loc}}(\Pi_2)$ -location indistinguishable against PPT adversaries, where $\mathsf{Adv}_{\mathcal{A}}^{\mathrm{Ide}}(\Pi_2) \leq \frac{1}{m^2 * 2^{|pi|+(z+1)*|K|}}$ when the target user is a rider or a driver.*

Proof: We also have two types of locations: rider's current location cl and destination dn inserted into an indistinguishable Bloom filter $\mathcal{IBF}$ and driver's sets of two hashes $\{(H_o(pr), H_{z+1}(H_o(pr)))\}_{o=1}^{z}$. They are also encrypted under the enclave's public key. Again, we proceed with the two types of adversaries, driver and rider, RSU and RHSP.

(1) Driver and rider as $\mathcal{A}$. If the target user is a rider, $\mathcal{A}$ tries to recompute the $\mathcal{IBF}$. $\mathcal{A}$ can correctly guess k by (1) correctly guessing the value of secret keys K_i, $1 \leq i \leq z+1$ and the two cell indexes or (2) randomly guessing the two cell indexes. As $\mathsf{Adv}_{\mathcal{A}}^{\mathrm{Loc}}(\Pi_2) = \Pr[k' = k] - \frac{1}{m^2}$, we have $\mathsf{Adv}_{\mathcal{A}}^{\mathrm{Loc}}(\Pi_2) = \Pr[k' = k] - \frac{1}{m^2} = \frac{n-1}{n} * \frac{1}{m^2} * \frac{1}{2^{|pi|}} * \frac{1}{2^{(z+1)*|K|}} + \frac{1}{m^2} - \frac{1}{m^2} = \frac{n-1}{n} * \frac{1}{m^2} * \frac{1}{2^{|pi|}} * \frac{1}{2^{(z+1)*|K|}} < \frac{1}{m^2 * 2^{|pi|+(z+1)*|K|}}$, which is negligible.

If the target user is a driver, $\mathcal{A}$ aims to recompute the $\{(H_o(pr), H_{z+1}(H_o(pr)))\}_{o=1}^{z}$. $\mathcal{A}$ can correctly guess k by (1) correctly guessing $\{K_i\}_{i=0}^{z}$ or (2) randomly guessing the two cell indexes. This makes the advantage the same as the one above, i.e., $\mathsf{Adv}_{\mathcal{A}}^{\mathrm{Loc}}(\Pi_2) < \frac{1}{m^2 * 2^{|pi|+(z+1)*|K|}}$.

(2) RSU and RHSP as $\mathcal{A}$. Similar to the rider and driver as $\mathcal{A}$, they aim to recompute $\mathcal{IBF}$ of a rider or $\{(H_o(pr), H_{z+1}(H_o(pr)))\}_{o=1}^{z}$ of a driver. They can correctly guess k by (1) correctly guessing $\{K_i\}_{i=0}^{z}$ or (2) randomly guessing the two cell indexes. We have $\mathsf{Adv}_{\mathcal{A}}^{\mathrm{Loc}}(\Pi_2) = \Pr[k' = k] - \frac{1}{m^2} = \frac{1}{m^2} * \frac{1}{2^{|pi|}} * \frac{1}{2^{(z+1)*|K|}} + \frac{1}{m^2} - \frac{1}{m^2} = \frac{1}{m^2} * \frac{1}{2^{|pi|}} * \frac{1}{2^{(z+1)*|K|}} = \frac{1}{m^2 * 2^{|pi|+(z+1)*|K|}}$, which is negligible.

To summarize, Π_2 is $\mathsf{Adv}_{\mathcal{A}}^{\mathrm{Loc}}(\Pi_2)$-location indistinguishable against PPT adversaries.

5.2 Security Level I

Theorem 4. *Our basic Nereus scheme Π_1 is $\mathsf{Adv}_{\mathcal{A}}^{\mathrm{Col}}(\Pi_1)$-location indistinguishable against colluding PPT adversaries, where $\mathsf{Adv}_{\mathcal{A}}^{\mathrm{Col}}(\Pi_1) = \frac{1}{2^{|g|+|pi|}}$.*

Proof: To successfully link the request of the matched rider from n' riders, a malicious driver who is matched to the rider, colludes with the $\mathcal{A}$ (RHSP, RSU) to share information. $\mathcal{A}$ can "corrupt" a driver to obtain information. The attack aims at leakage from both identity and location such that $\mathcal{A}$ can correctly guess k by (1) correctly guessing the value of u of g^u belonging to the group secret key gsk, the value of private key pi^{UM} of the enclave, and the two cell indexes or (2) randomly guessing k. Since the RHSP (RSU) has colluded with the driver, it learns the two cell indexes of the matched rider. Even though there is a collusion between the driver and the RHSP (RSU), the SGX-protected secure matching environment has successfully cut off the operability from the traditional handler, i.e., traditional RHSP (RSU). It leaves no possibility for the colluding parties to acquire any useful information on user matching. As $\mathsf{Adv}_{\mathcal{A}}^{\mathrm{Col}}(\Pi_1) = \Pr[k' = k] - \frac{1}{n'}$, we have $\mathsf{Adv}_{\mathcal{A}}^{\mathrm{Col}}(\Pi_1) = \Pr[k' = k] - \frac{1}{n'} = \frac{1}{2^{|g|}} * \frac{1}{2^{|pi|}} + \frac{1}{n'} - \frac{1}{n'} = \frac{1}{2^{|g|}} * \frac{1}{2^{|pi|}} = \frac{1}{2^{|g|+|pi|}}$, which is negligible.

The SGX enclave is a secure execution environment safe from the RSU. Even though the pseudonym and cellphone number are decrypted in the enclave, the RSU cannot access it. Therefore, the user profiling attack does not work. To summarize, the basic Nereus scheme is $\mathsf{Adv}_{\mathcal{A}}^{\mathrm{Col}}(\Pi_1)$-collusion resistant against colluding PPT adversaries.

Theorem 5. *Our advanced Nereus scheme Π_2 is $\mathsf{Adv}_{\mathcal{A}}^{\mathrm{Col}}(\Pi_2)$ -location indistinguishable against colluding PPT adversaries, where $\mathsf{Adv}_{\mathcal{A}}^{\mathrm{Col}}(\Pi_2) = \frac{1}{2^{|g|+|pi|}}$.*

Proof: The proof resemble the one for the basic Nereus scheme, except that $\mathcal{A}$ can correctly guess k by (1) correctly guessing the value of u of g^u, the value of gsk, the value of pi^{UM}, the two cell indexes, and $\{K_i\}_{i=0}^{z}$ or (2) randomly guessing k. After collusion, $\mathcal{A}$ can have the two cell indexes and $\{K_i\}_{i=0}^{z}$. We have $\mathsf{Adv}_{\mathcal{A}}^{\mathrm{Col}}(\Pi_2) = \Pr[k' = k] - \frac{1}{n'} = \frac{1}{2^{|g|}} * \frac{1}{2^{|pi|}} + \frac{1}{n'} - \frac{1}{n'} = \frac{1}{2^{|g|+|pi|}}$, which is negligible.

To summarize, the advanced Nereus scheme is $\mathsf{Adv}_{\mathcal{A}}^{\mathrm{Col}}(\Pi)$ -collusion resistant against colluding PPT adversaries.

5.3 Security Level II

In the advanced protocol, we have designed a verification process and a commitment process. Riders and drivers are required to put some deposits as a commitment to their next ride. The matched users have to provide valid inputs to their earlier commitments as well as their request (response) to the matched user to show their qualification of the ride.

During proof verification, users' commitments are recomputed, and the RVSC verifies requests (responses). If a malicious user has submitted different data to their matched user to profit unfairly, it will be detected immediately by the SC. After the requests and responses are recorded on the blockchain, users can verify whether their received matching result is valid and the best locally. It is achieved by inserting the response indexes into the request IBF. Finally, users can verify the returned matching result.

For the users who sent false data, the Transfer function will be called to transfer the deposit to the other user and punish the misbehaving user. In addition, if a malicious user abandons the matched user, the victim calls the Complain function to claim compensation. The deposit commitment and PoR prevent malicious drivers from not committing to their responses. Eventually, runaway users are punished. Therefore, Nereus guarantees matching verifiability and accountability.

We compare Nereus with existing privacy-preserving RHSs regarding security and privacy in Table 2. For [9], we choose its DAP-DAD (determined area for pick up and determined area for drop off) sub-scheme to compare because it shares the same ride-hailing case with ours. It uses group signatures (not mentioning which one) to protect identity privacy, authentication, accountability. It protects the locations of drivers and rider by encrypting them into keyword index and query token. pRide [10] focuses on protecting location privacy. They convert the original road

TABLE 2
Comparison of Security and Privacy Properties

Property	Identity Privacy	Location Privacy	Collusion Resistance	Verifiability	Accountability	Authentication	TTP Required
DAP-DAD [9]	√	√			√	√	√
pRide [10]		√					√
FICA [11]	√	√			√	√	√
CoRide [12]	√	√			√	√	√
B-Ride [13]	√	Partial			√	√	Partial
Nereus Basic	√	√	√			√	√
Nereus Advanced	√	√	√	√	√	√	√

networks into a higher dimensional space to improve efficiency. They leverage homomorphic encryption and garbled circuit to compare distances. FICA [11] provides identity privacy, authentication, and accountability by using anonymous credentials. It protects locations by encrypting them into Bloom filter-based index and query token. CoRide [12] protects identity privacy based on Zerocash and matches location by leveraging indistinguishable Bloom filter and query tokens. B-Ride [13] offers partial location privacy due to the use of spatial cloaking. Such a technique is not secure due to the correlating attack [40]. We mark "Partial" in "TTP Required" for B-Ride because it requires not a certificate authority but a location prover to ensure the authenticity of the reported location sent by the drivers.

Discussions. The rating for the order is submitted by the driver, but the rating is generated and signed by the rider. So the rider can see the rating in the ride complete transaction and verify it in case of malicious modification from the driver. The RHSP does not hold the secret to decrypt user data. Users can query their previous ride data from the RHSP or pass the temporary private key to an investigator to retrieve and decrypt the stored data on RHSP. In offline dictionary attacks, an RSU or an adversary tries to guess a rider's location because the number of cells limits them. Nereus can resist this attack because we have used secret keys to protect the cell index. When the RSU colludes with a driver, there are two cases: (1) The driver is not the one matched to the target rider, the driver and RSU cannot recover the request from its encrypted form er; (2) The driver is the one matched to the target rider, they can only know which cell index the rider is in before negotiation. In the end, the driver has to know the exact location of the rider to pick her up.

6 Performance Analysis

6.1 Experimental Settings

We implemented a prototype of Nereus using Ethereum and Intel SGX. We use a desktop (Lenovo Thinkpad X1 Carbon) as RSU, with 8GB RAM and a SGX supported CPU (i5-10210u, 4-core, base frequency 1.60GHz, max frequency 4.20GHz). We list experimental parameters in Table 3 and upload source codes to https://github.com/UbiPLab/Nereus.

We install geth (Ethereum client) on our RSU node to communicate with the other three computers (blockchain nodes) and deploy the MMSC contract. We use `puppeth` to create the genesis block with Proof-of-Authority consensus mechanism (Clique). Both SGX-protected EVM and geth client exchange data with a middle layer (Python or other language program) via `web3rpc`. For geth side contracts deploying, we choose metamask which is a light-weighted browser plugin. We use Java (JDK15) to implement cryptographic primitives and use Python language as the middle layer to simplify communication operations.

6.2 SGX Environment Configuration

To prepare the SGX runtime and development environment, we clone the SGX source codes published by Intel and install the required tools. Then we build the project to generate SGX-SDK and SGX Platform Software (PSW) installer. Finally, we install them through `deb` package tool. It takes 8 ms to seal data of 1 KB in the enclave as recorded in Table 4.

6.3 SGX-protected EVM Configuration

To deploy an UMSC and an RVSC within a SGX enclave environment, we introduce a framework enclave Ethereum Virtual Machine (EVM) released by Microsoft. First, we compile our UMSC into bytecode and export application binary interface information by using `Solidity Compiler` as a preprocess of SCs. Next, we setup Open-enclave environment and prepare a Confidential Consortium Framework (CCF). Afterward, we refer to EVM-for-CCF repository provided by Microsoft to integrate enclave EVM with CCF. To build the projects mentioned above, we develop on Ubuntu 18.04 and compiling with `Clang8-`. After the environment is built, we deploy contracts, submit transactions, and get transaction information through `web3` communication.

6.4 Smart Contract Deployment

To simplify the development of SCs, we use Remix (a browser-side SC development tool) to write codes. We choose Ganache as the basic function test tool for SCs in a none-SGX environment. After developing SCs, we use solc to compile the SCs into bytecode files. For MMSC deployment, we

TABLE 3
Experimental Parameters

Parameter	Value	Parameter	Value
m	1024	l	8000
z	10	k	11
N	32	$\|K\|$	256
$\|pk\|$	1296	$\|pi\|$	5088
$\|p\|$	160	$\|g\|$	1024

TABLE 4
Performance Benchmarks of SGX Operations

Operation	Time Cost (Millisecond)
Enclave Creation	45
Sealing, Unsealing (1 KBytes)	8, 7

choose truffle, which is a javascript based Ethereum development framework. While for UMSC deployment, EVM-for-CCF provides some Python tools to read bytecode files and deploy them in SGX-protected EVM via `web3rpc`. Contract state is only visible to the executing nodes.

6.5 Gas Costs and Monetary Costs

The monetary costs come from the gas costs of operating SCs in Ethereum. In the basic protocol, a request transaction and a response transaction are submitted. Next, a basic match transaction is sent to match users. The test script was written in Python 3.7 with web3py library, and we collect the GasUsed value through transaction receipts. We repeated the experiment for 1 to 10 ride requests and obtained the gas costs for each function in the SC. The ether price is 1,957 on 02/21/2021 and the gas price is 1 Gwei (10^{-9} ether, 1 ether equals to $1,957) (source: https://coinmarketcap.com). The gas costs are shown in Fig. 5 where it only costs $0.07 and $0.19 to match users in basic protocol and advanced protocol, respectively. Calling `Request()` dominates the gas costs due to the Bloom filter.

Since we build our RHS system upon Ethereum, both riders and drivers will pay transaction fees and may suffer a varying fee because of ether price's unstable nature. We can alleviate this problem by building a consortium blockchain among ride-hailing companies, insurance companies, transportation departments, and other vehicle-related institutions [59], [60]. Using the consortium blockchain, the miners/stakeholders can set the transaction fee to a fixed and acceptable value. There are already some industrial adoptions of blockchain in RHS, such as MOBI [37], DRIFE [38], and TADA [39]. They lay a foundation for further collaboration on consortium blockchain-based RHS.

6.6 Computational Costs

We now analyze the computational costs for three entities: rider, driver, and RSU through counting the cryptographic operations and recording the measured time in Table 5. In the basic protocol, a rider's computational cost rests in ride requesting, and ride initiation and complete. It consists of

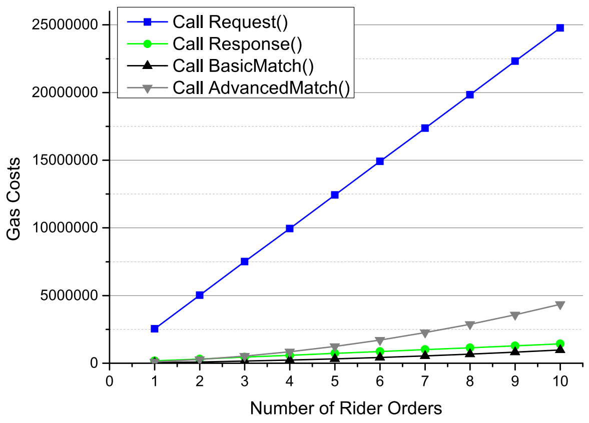


Fig. 5. Monetary Costs.

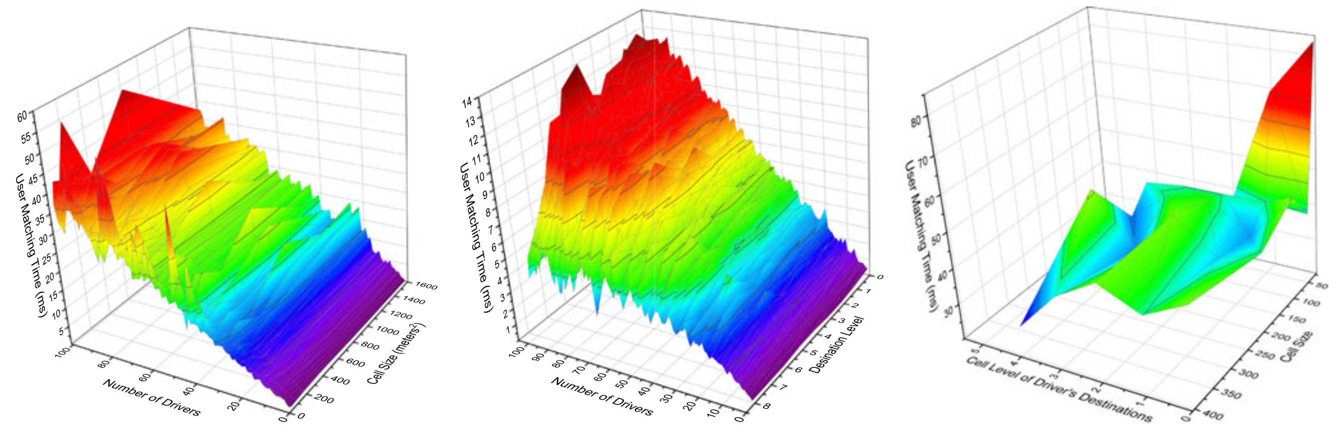
$2 * k$ hashings of current location cl and destination dn, computing a pseudonym pn, an asymmetric encryption E of ride request, generating three signatures in Tx^{R}_{Req}, Tx^{D}_{Ini}, and Tx^{D}_{Com}, and encrypting data for cloud storage. A driver encrypts a ride response, generates three signatures, and encrypts data. An RSU decrypts the ride request and response, hashes driver's request into rider's Bloom filter, encrypts two matching results, and generates two signatures.

In the advanced protocol, we introduce privacy-preserving range query and commitment, which cause $2 * z * |\mathcal{PF}| + 1$ hashes for rider, $4 * z * |\mathcal{MS}| + 1$ hashes for driver, $2 * z * |\mathcal{MS}|$ hashes in matching for RSU, and 2 hashes in verifying for RSU as well as generating Merkle proofs. The computational costs of users are promising, i.e., 24 ms/36 ms for basic/advanced rider and 25 ms/32 ms for basic/advanced rider. This is due to lightweight hashing and one asymmetric encryption. An RSU has to consume 200 ms/196 ms due to SGX operations. The implemented running time are recorded in Table 5.

For the advanced protocol where we use secure range query to match users, we evaluate the effectiveness of this method in detail. Specifically, we take the status of all the drivers (i.e., number of available drivers n, the cell size c, the drivers' destination level d) in the system into consideration and conduct three sets of experiments to record the average user matching time. Each set of experiment has two of the three variables n, c, d. We arrange square cells into different levels, and each cell in the upper levels covers four levels on the level below. The level on top is called level 1. The map is represented as an $k \times k$ matrix, so a cell can be encoded as a string with a length $2log(k)$, and each expansion will occupy 2 bits of the code. For example, $k = 32$,

TABLE 5
Performance Analysis and Comparison with Existing Work

	Computational Costs (Millisecond)			Communication Cost (KBytes)		
Entity	Rider	Driver	RSU/RHSP	Rider	Driver	RSU/RHSP
DAP-DAD [9]	502	406	1.5	15690	15688	2.91
*p*Ride [10]	3600	3600	137000	2	2	8700
FICA [11]	163	160	202	0.68	1.00	2.76
CoRide [12]	144	138	162	2.53	1.13	0.87
B-Ride [13]	182	251	107	0.84	2.34	0.19
Nereus Basic	24	25	62	1.22	1.90	1.14
Nereus Advanced	36	32	127	1.25	1.92	2.01



(a) Advanced-Set 1 (varying n and c) (b) Advanced-Set 2 (varying n and d) (c) Advanced-Set 3 (varying c and d)

Fig. 6. Time of user matching of the advanced protocol.

the coordinate of a selected cell is (14,195) and its encoded string is 0110101010100010, its expansions will be $(01101010101000**, 011010101010****, 0110101010*****{*}, \ldots, ***************{*})$. We set the drivers' current location on the level above the lowest level to control the distance of pick-up riders. If a driver's destination is on the highest level, it means the driver can go anywhere in the city. A destination on the lowest level 0 means the driver only wheels to a small region. For n, we choose from [1,100]. For c on the lowest level, we set cell size (meter^2) within $[25^2, 1600^2]$. For d, we randomly put the drivers' destination in the cells of the lowest level and elevate them to upper levels accordingly. The number of riders is 10.

From Figs. 6a and 6b, the average user matching time increases with n when c and d are fixed. In the third set of experiment, we set the number of divers to be 100. The biggest cell size is 400. Since we have to expand the level of drivers' destinations, if the cell size on the level 0 is too big, then we cannot expand it. From Fig. 6c, when c becomes bigger, it is easier for a driver's cell to cover more riders, thus reducing the matching time. When drivers' destination cell level is lifted, their cell will cover more cells in the lowest level, thus increasing drivers' chance of taking rider orders and reducing the whole matching time. The effects of c and d are also shown in Fig. 6a, but it is not that obvious due to the dominance of driver number. The odd points exist because there are some cases when matching process requires more time if the target driver is at the end of the searching list.

We also conduct experiments on parallel processing for multiple riding requests. We treat the drivers' response queue as sharing resource and set 4 (number of CPU threads) threads to process riders' requests, which are assigned to the threads equally. To reduce thread conflict, we adopt the *CopyOnWrite* mechanism to optimize the queue and improve the matching speed. We use a counter to wait for all threads to finish and compute the time cost. Next, we evaluate the maximum, minimum, and average matching time under the multi-user scenario. There are twenty sets of experiments where the numbers of riders are drawn from [10,200]. We fix the number of drivers as 200 to simulate the sufficiency of available drivers. The results in Fig. 7 show that the consumed time of the parallel processing increases with the number of requests, and it is less than 450 ms when there are 200 requests.

6.7 Communication Overhead

In the basic protocol, a rider sends a ride request transaction Tx_R^{Req} to the blockchain network, two signatures to the driver, and encrypted data ed^R to RHSP, i.e., $|\text{"Request"}| + |U| + |er_R| + |ts_R^{\text{Req}}| + |\sigma_R^{\text{Req}}| + |\sigma_R^{\text{Ini}}| + |\sigma_R^{\text{Com}}| + |ed_R| = 7*8 + 10 + 512*8 + 8*8 + (2*160 + 2*160) + 560*2 + 512*8 = 9974$ bits $= 1.22$ KBytes. A driver sends a ride response transaction Tx_D^{Res}, a ride initiation transaction Tx_D^{Ini}, a ride complete transaction Tx_D^{Com}, and encrypted data ed_D, i.e., 1.90 KBytes. An RSU returns two encrypted matching results mr_U^R, mr_U^D, and two signatures $\sigma_{\text{UM}}^R, \sigma_{\text{UM}}^D$ to rider and driver, i.e., 1.14 KBytes. In the advanced protocol, a rider/driver sends additionally an value d and a commitment com^R in ride request/response. In ride verifying, the rider/driver sends a ciphertext to the RSU. The RSU returns Merkle proofs, i.e., $log_2 N + 1$ hashes, to the rider/driver. In ride initiation and complete, the rider and driver send similar messages as in the basic protocol, except that the new initiation transaction Tx_{Ini}^D includes $(IBF_R, rn_R, rr_D(1), Mp_R, Mp_D)$.

6.8 Comparison with Existing Work

DAP-DAD [9], pRide [10], FICA [11], CoRide [12], and B-Ride [13] are constructed upon cryptographic operations. DAP-DAD [9] utilizes privacy-preserving multi-keyword ranked search over encrypted data to match rides with drivers. The users have to compute a large-sized index which leads to higher cost, and the server has a lower cost for only multiplying the two indices by using inner product. pRide [10]'s costs are high for using homomorphic encryption and Yao's garbled circuit for the distance comparisons. FICA also utilizes RSUs as fog nodes to locally match riders and drivers by using anonymous authentication, private proximity test, and private range query. Its high computational costs are caused by bilinear pairings. CoRide integrates anonymous authentication, private proximity test and privacy-preserving query processing. Its performance is degraded by using bilinear pairings and Zerocash. B-Ride uses zero-knowledge set membership to prove that the driver has arrived at the current location. A performance

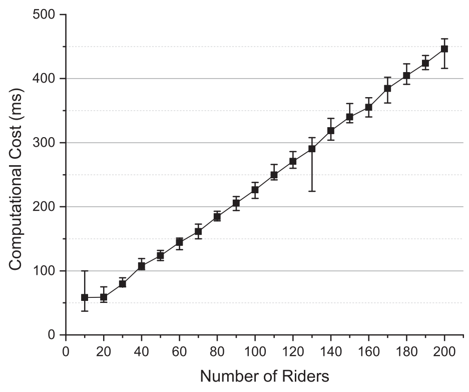


Fig. 7. Computational costs of parallel processing.

penalty is incurred by (asymmetrically) decrypting ride responses and proving a location. From Table 5, Nereus achieves a performance improvement in computational costs and has a reasonable communication overhead.

7 Discussions

7.1 Balancing Privacy and Efficiency

In this work, we have used blockchain and SCs for user matching in RHS while addressing privacy issues. A consequent concern is a degradation in performance. It is mostly raised by the service providers that put the performance first, compared to the user privacy. However, we believe that privacy preservation is important to users in RHS, and there should be a balance between privacy and performance. First, we know that users' awareness toward privacy is increasing these years, according to reports [41], [42], European Union [43], and academic research [44]. To further validate this observation, we initiated a ballot on the Amazon Mechanic Turk, including a question "Now you are a rider using a ride-hailing app, are you willing to protect your privacy (identity and location) but at the cost of degrading a little system performance?" with three answers being "Yes", "No", and "Not sure". From February 21, 2021, to June 18, 2021, we collected 554 answers from 554 workers. Among them, 372 workers (67.1% of all) chose "Yes", agreeing with our assumption. The result highlights the need to raise privacy awareness and construct a privacy-preserving RHS. Second, the experimental results show that it is possible to preserve privacy while only minimally impacting system performance, achieving a good balance between the two aspects.

7.2 Combining Technologies

We have combined several techniques in Nereus, especially Ethereum blockchain, SC, and SGX. It requires great efforts to dynamically combine them into an organic whole, e.g., integrating SCs into SGX enclaves. For instance, if we append SGX to the blockchain, user matching and SC are separated. In other words, the user matching on RSU is carried out off-chain. Such a method has two consequences. First, extra communication is incurred between the RSU and the blockchain. Second, it will require more efforts to update the SC in the future. What we do is put the SCs into SGX enclaves. It comes with a complicated technical challenge: the system dependency problem. When establishing the SGX environment, we must detect whether the CPU supports SGX with Flexible Launch Control (FLC) that will affect the subsequent process. For EVM-for-CCF on Ubuntu18.04, `llvm` toolchain and `clang8` environment are necessary, but it is difficult to install them. The CCF submodule that comes with EVM-for-CCF is inconsistent with the official source CCF.

Although Intel SGX support has been discontinued from 11th generation (or newer) desktop processors, some new processors with Intel server platform services are still applicable to our solution, e.g., the Intel Xeon D-1702 Processor [45] released in 2022 with the lowest price among processors of the same type. Intel also used Xeon series processors in a 5G-RSU solution [46]. Therefore, we believe that the deploy-ability of our system is guaranteed even without desktop processors.

7.3 Cells for Matching

The rider's destination must fall in at least one drivers cell to achieve a ride matching. At first sight, it may be difficult for the driver to choose from many cells, which causes the rider to wait for the responses from drivers. We solve this problem by utilizing a secure range query [35] and cells of different levels to match users. If a driver's destination cell level is higher than the one of a rider's and the driver's cell covers the rider's cell, their destinations can be matched since the index of a lower level cell belongs to the range of a higher level cell.

7.4 Side-Channel Information

We follow the "encrypt-decrypt-mix-match-encrypt" principle to match requests and responses. When the driver meets the rider, the colluding service provider and driver cannot link a rider to her/his request. However, there are some side-channel information for the colluding parties to locate the driver. (1) Unique communication patterns of each smart phone. Since the rider communicates with the ride-hailing service via HTTP, side-channel information, such as operating system and version of an internal browser (possibly invoked by the RHS app) are exposed. At first sight, the adversary can leverage these information to link the same rider. However, such information can be removed before sending to the RSU to eliminate concerns. Even if they are observable, which notifies the adversaries of a previous rider has sent a request, we still protect the link between the rider and the rider's request based on our principle. In other words, the adversaries still cannot know which request traces back to which rider. (2) IP addresses. We can use the SOCKS protocol [47] to exchange network packets between a rider and an RSU through a proxy server. (3) Request-response time: We assume that there are multiple requests in an area, which leads to basic k-anonymity for riders. Meanwhile, we can help riders exclude a previously-matched driver and reduce the privacy risk of being recognized. We have made an effort to achieve this goal in our previous work [48].

7.5 Multiple and Anonymous Credentials

Multiple credentials allow the riders to use a fresh credential every time. Anonymous credentials hide the rider's credential among a group of credentials. For instance, decentralized anonymous credentials [49] enable users to prove their identities with strong privacy guarantees without relying on trusted parties. A user Alice issues a credential by sending a credential (commitment) c, a pseudonym p, some attributes $\{att_i\}$, and a proof stating that c and p contain the same secret key sk and $\{att_i\}$ are in some allowed set. Alice shows c by proving that she knows a credential on the ledger, c opens to the same sk, and it has some attributes. *However, the two approaches do not work under our observed collusion attack. This roots in the fact that the NSP will always learn which anonymous credential is matched to its colluding driver and then link the credential to the actual rider. Therefore, the credentials are no longer anonymous.*

7.6 Reputation and Blacklist-Based Mechanism

Establishing a reputation-based and anonymous evaluation for the drivers in the matching process and setting up a blacklist for malicious drivers can regulate drivers' behaviors to some extent and prevent potential attacks on riders. In the Reputation and Blacklist-based Mechanism (RBM), it is still possible that drivers misbehave at the cost of reduced reputation. For example, an Uber driver assaulted a sleeping rider [50], and an Uber driver abused a woman passenger [51], even if Uber has adopted a rating system [52]. Although the RBM is not perfect, we can adopt it as a supplement to our design. Furthermore, the RBM will be more appealing if it is linked to the social credit system and multiple financial systems. However, we design the "encrypt-decrypt-mix-match-encrypt" principle to defend against the collusion attack, which is different from the RBM. Besides, the RBM itself is a complex system that calls for thoughtful designs. For instance, it is fair to apply a two-way rating mechanism, i.e., a rider and a driver can rate each other after a ride. It is better to utilize zero-knowledge proofs to verify a rider's reputation since a time-varying reputation value can track a rider. Last, it is practical to adopt privacy-preserving rating techniques for riders. It means that the riders stay anonymous after rating, i.e., updating reputation anonymously. Although we have used a simple rating mechanism in Nereus, we could design a complex-but-appropriate RBM for RHS and insert it into our matching processing in future work.

7.7 Security Concerns and Deployment of Trusted Hardware

There have been many defense methods for practical attacks on the trusted hardware. For example, program obfuscation for branch prediction attack [53], modifying kernel's memory management service for memory attack [54], partitioning the set associative memory structures for contention-based and access-based cache attacks [55], and padding executing time to a constant by dummy loop for the timing-channel attack [56].

Our design requires deploying trusted hardware in every RSU and is scalable enough for real deployments for some reason. (1) Privacy. There is a strong need for privacy protection given the conflicts and crimes in RHSs. The deployment of SGX-protected matching process on each RSU will help the RHS entrepreneurs reduce riders' privacy concerns and improve their user viscosity for final profiting. (2) Cost. The deployment of trusted hardware in every RSU does the cost for the RHS company. However, such a cost is relatively small compared to the company revenue. For example, the unit price for an SGX-supported chip is \$42 [57], and the cost of updating 100 RSUs in each of 100 cities with such a chip is \$420,000. According to an official report [58], Uber's gross bookings reached an all-time high of \$23.1 billion in the third quarter of 2021. Hence, the cost is not a significant impediment to broad deployment.

7.8 Blockchain and TTP

Integrating a blockchain to RHS makes the ride records difficult to be tampered with (by the RHSP) and easy to be audited (by an arbiter or a police investigator [59]). Blockchain has been adopted in vehicular schemes, such as decentralized and automated incentives for price auditing in RHSs [61] and anonymous and secure vehicular digital forensics based on blockchain [59]. At first glance, the TTP is contradictory to our decentralized framework. However, the TTP is responsible for generating public parameters and cryptographic keys. Once the RHS is running, the TTP stays offline. Notably, the TTP is reserved for security and privacy concerns. If some users misbehave, the TTP can reveal their identity [34].

Maram et al. proposed CANDID, a pure blockchain-based system without TTP [62], to build a platform for the practical and user-friendly realization of decentralized identity. They claim that decentralized identity witnesses a huge risk of key loss. Existing approaches assume the spontaneous availability of a TTP and raise a bootstrapping problem. They also lack resistance to Sybil attacks and the ability to detect misbehaving or sanctioned users while preserving user privacy. CanDID issues credentials that draw securely and privately on data from existing, unmodified web service providers such that users can leverage their online accounts to recover lost keys. CanDID offers confidentiality for keys, identities, and data, and allows identification by using a decentralized committee of nodes. However, our focus is to defend against the collusion attack, which is different from CANDID. A TTP-free solution is possible for RHS. From privacy perspective, it removes the privacy concerns raised by the centralized collection of users' information. Uses do not have to submit their identity information to the TTP during registration. From function perspective, it requires different techniques to match riders and drivers, i.e., how to securely match riders and drivers without a key distribution step. From efficiency perspective, it needs a good balance between secure matching and querying efficiency. From accountability perspective, it poses a challenge on how to reveal the real identity of a malicious user in a decentralized and anonymous network without a TTP.

7.9 Edge Computing

We leverage edge computing to shift the user matching from the central RHSP to distributed RSUs to improve matching efficiency. Edge computing enables computation by edge nodes, e.g., RSUs, at the network edge, on downstream data on behalf of cloud services and upstream data on behalf of IoT services. It has been adopted in vehicular networks, such as real-time navigation [63], parking spots sharing [64], and traffic monitoring [65]. We have also conducted efficiency experiments to show the advantage of edge computing [65].

7.10 Why Existing Work Cannot Solve the Collusion Attack

Li et al. [66] consider a collusion attack between the matching executor and a driver, where the executor assigns its accomplice driver to riders instead of assigning more suitable drivers. Their solution is to ask the executor to prove that (1) the assigned driver is the best candidate for the next rider, (2) the driver is taking a rider, and (3) the driver is taking an authenticated rider. Although they solved an important problem in RHS, still the executor and driver can link the anonymous riders and their anonymous request. Alshammari et al. [67] consider a collusion attack, i.e., collusive malicious feedback behaviors, where a set of people acting in concert gives false feedback to decrease or increase the productivity of ratings of products on an e-commerce site. However, this attack is uploading false data by colluding users. They defend against it by designing a trust algorithm for detecting malicious behaviors by calculating three criteria: malicious recommendations detection, malicious recommenders' behavior, and collusion attack frequency. Ugur [68] presented a recommendation-based manipulator attack on the Social Internet of Things. Collaborated devices execute the attack by exploiting bullet-staff countermeasure approaches to eliminate the honest devices and take over the service as a provider. Thus, the last two attacks are totally different from our setting, and their solutions do not apply to our collusion attack.

8 Conclusion

We have presented Nereus to realize private RHS with SGX-protected SCs. With Nereus, the new collusion attack can no longer violate the identity privacy of riders. Riders and drivers do not have to perform heavy cryptographic operations. The RSU can securely and efficiently match users. We have formally defined and analyzed the security and privacy of Nereus, including the capability to withstand collusion attacks. We implement a prototype of Nereus based on Ethereum and SGX. Experimental results indicate that Nereus exhibits a significant performance improvement over existing schemes. Our contributions advance the state of the art in rider identity privacy protection against collusion attack for RHSs. The analysis results also indicate a promising solution to other privacy-preserving applications in vehicular networks.

Acknowledgments

We sincerely thank Prof. Zijian Zhang from Beijing Institute of Technology (BIT), China for discussions on the cryptographic games.

References

[1] S. Little, "Uber data leak hit 2.7m UK customers, admits ride-hailing company," Independent, 2017. Accessed: Sep. 05, 2021. [Online]. Available: https://www.independent.co.uk/news/business/news/uber-data-leak-customers-ride-hailing-company-taxi-app-london-ban-a8082566.html

[2] A. Pham et al., "Privateride: A privacy-enhanced ride-hailing service," *Proc. Privacy Enhancing Technol.*, vol. 2, pp. 38–56, Apr. 2017.

[3] L. Zhu, M. Li, Z. Zhang, and Z. Qin, "ASAP: An anonymous smart-parking and payment scheme in vehicular networks," *IEEE Trans. Dependable Secure Comput.*, vol. 17, no. 4, pp. 703–715, Jul./Aug. 2020, doi: 10.1109/TDSC.2018.2850780.

[4] M. Li, Y. Chen, S. Zheng, D. Hu, C. Lal, and M. Conti, "Privacy-preserving navigation supporting similar queries in vehicular networks," *IEEE Trans. Dependable Secure Comput.*, vol. 19, no. 2, pp. 1133–1148, Mar./Apr. 2022, doi: 100.1109/TDSC.2020.3017534.

[5] Y. Xiao, N. Zhang, J. Li, W. Lou, and Y. T. Hou, "PrivacyGuard: Enforcing private data usage control with blockchain and attested off-chain contract execution," in *Proc. 25rd Eur. Symp. Res. Comput. Secur.*, 2020, pp. 610–629.

[6] Z. Zhang, Z. Qin, L. Zhu, J. Weng, and K. Ren, "Cost-friendly differential privacy for smart meters: Exploiting the dual roles of the noise," *IEEE Trans. Smart Grid*, vol. 8, no. 2, pp. 619–626, Mar. 2017.

[7] J. He, Z. Zhang, J. Mao, L. Ma, and L. Zhu, "Video aficionado: We know what you are watching," *IEEE Trans. Mobile Comput.*, vol. 21, no. 8, pp. 3041–3052, Aug. 2022.

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

[9] 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, Feb. 2017.

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

[11] M. Li, L. Zhu, and X. Lin, "Efficient and privacy-preserving carpooling using blockchain-assisted vehicular fog computing," *IEEE Internet Things J.*, vol. 6, no. 3, pp. 4573–4584, Jun. 2019, doi: 10.1109/JIOT.2018.2868076.

[12] M. Li, L. Zhu, and X. Lin, "CoRide: A privacy-preserving collaborative-ride hailing service using blockchain-assisted vehicular fog computing," in *Proc. 15th ACM EAI Int. Conf. Secur. Privacy Commun. Netw.*, 2019, pp. 408–422.

[13] M. Baza, N. Lasla, M. Mahmoud, G. Srivastava, and Mohamed Abdallah, "B-Ride: Ride sharing with privacy-preservation, trust and fair payment atop public blockchain," *IEEE Trans. Netw. Sci. Eng.*, vol. 8, no. 2, pp. 1214–1229, Apr.–Jun. 2021.

[14] "Didi: China blocks ride-hailing app over privacy breach," 2021. Accessed: May 01, 2022. [Online]. Available: https://www.dw.com/en/didi-china-blocks-ride-hailing-app-over-privacy-breach/a-58156362

[15] K. Mueffelmann, "Ubers privacy woes should serve as a cautionary tale for all companies," 2015. Accessed: May 02, 2022. [Online]. Available: https://www.wired.com/insights/2015/01/uber-privacy-woes-cautionary-tale

[16] M. Li, N. Cao, S. Yu, and W. Lou, "FindU: Privacy-preserving personal profile matching in mobile social networks," in *Proc. IEEE 30th Int. Conf. Comput. Commun.*, 2011, pp. 2435–2443.

[17] R. Zhang, J. Zhang, Y. Zhang, J. Sun, and G. Yan, "Privacy-preserving profile matching for proximity-based mobile social networking," *IEEE J. Sel. Areas Commun.*, vol. 31, no. 9, pp. 656–668, Sep. 2013.

[18] X. Yi, E. Bertino, F.-Y. Rao, and A. Bouguettaya, "Practical privacy-preserving user profile matching in social networks," in *Proc. IEEE 32nd Int. Conf. Data Eng.*, 2016, pp. 373–384.

[19] O. Pereira and R. L. Rivest, "Marked mix-nets," in *Proc. 21st Int. Conf. Financial Cryptogr. Data Secur.*, 2017, pp. 353–369.

[20] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 2009, Accessed: Sep. 05, 2021. [Online]. Available: https://bitcoin.org/bitcoin.pdf

[21] G. Wood, "Ethereum: A secure decentralised generalised transaction ledger," *Petersburg Version d77a387*, 2022, Accessed: May 02, 2022. [Online]. Available: https://ethereum.github.io/yellowpaper/paper.pdf

[22] F. McKeen et al., "Innovative instructions and software model for isolated execution," in *Proc. 2nd Int. Workshop Hardware Architectural Support Secur. Privacy*, 2013, pp. 1–8.

[23] I. Anati, S. Gueron, S. Johnson, and V. Scarlata, "Innovative technology for CPU based attestation and sealing," in *Proc. 2nd Int. Workshop Hardware Architectural Support Secur. Privacy*, 2013, pp. 1–7.

[24] Intel, "Intel ® software guard extensions (Intel ® SGX) developer guide," 2022. Accessed: Sep. 05, 2021. [Online]. Available: https://software.intel.com/content/www/cn/zh/develop/download/intel-software-guard-extensions-intel-sgx-developer-guide.html

[25] M. J. Freedman, K. Nissim, and B. Pinkas, "Efficient private matching and set intersection," in *Proc. 21st Int. Conf. Theory Appl. Cryptogr. Techn.*, 2004, pp. 1–19.

[26] A. Kosba, A. Miller, E. Shi, Z. Wen, and C. Papamanthou, “Hawk: The blockchain model of cryptography and privacy-preserving smart contracts,” in *Proc. IEEE 37th Symp. Secur. Privacy*, 2016, pp. 839–858.

[27] J. Guarnizo and P. Szalachowski, “PDFS: Practical data feed service for smart contracts,” in *Proc. 24h Eur. Symp. Res. Comput. Secur.*, 2019, pp. 767–789.

[28] S. Dziembowski, L. Eckey, and S. Faust, “FairSwap: How to fairly exchange digital goods,” in *Proc. 26th ACM SIGSAC Conf. Comput. Commun. Secur.*, 2019, pp. 967–984.

[29] F. Schuster et al., “VC3: Trustworthy data analytics in the cloud using SGX,” in *Proc. IEEE 36th Symp. Secur. Privacy*, 2015, pp. 38–54.

[30] W. Dai, C. Dai, K.-K. R. Choo, C. Cui, D. Zou, and H. Jin, “SDTE: A secure blockchain-based data trading ecosystem,” *IEEE Trans. Informat. Forensics Secur.*, vol. 15, pp. 725–737, 2020.

[31] S. Li, K. Xue, D. S. L. Wei, H. Yue, N. Yu, and P. Hong, “SecGrid: A secure and efficient SGX-enabled smart grid system with rich functionalities,” *IEEE Trans. Informat. Forensics Secur.*, vol. 15, pp. 1318–1330, 2020.

[32] M. Cebe, E. Erdin, K. Akkaya, H. Aksu, and S. Uluagac, “Block4Forensic: An integrated lightweight blockchain framework for forensics applications of connected vehicles,” *IEEE Comm. Mag.*, vol. 56, no. 10, pp. 50–57, Oct. 2018.

[33] R. Bost, B. Minaud, and O. Ohrimenko, “Forward and backward private searchable encryption from constrained cryptographic primitives,” in *Proc. 24th ACM SIGSAC Conf. Comput. Commun. Secur.*, 2017, pp. 1465–1482.

[34] D. Pointcheval and O. Sanders, “Short randomizable signature,” in *Proc. Cryptographers’ Track RSA Conf.*, 2016, pp. 111–126.

[35] R. Li and A. X. Liu, “Adaptively secure conjunctive query processing over encrypted data for cloud computing,” in *Proc. IEEE 33rd Int. Conf. Data Eng.*, 2017, pp. 697–708.

[36] M. Li, D. Hu, C. Lal, M. Conti, and Z. Zhang, “Blockchain-enabled secure energy trading with verifiable fairness in Industrial Internet of Things,” *IEEE Trans. Ind. Inform.*, vol. 16, no. 10, pp. 6564–6574, Oct. 2020, doi: 10.1109/TII.2020.2974537.

[37] “Mobility Open Blockchain Initiative,” 2022. [Online]. Available: https://dlt.mobi

[38] “DRIFE-Taxi 3.0 Ride-hailing reimagined,” 2022. [Online]. Available: https://www.drife.io

[39] “TADA,” 2022. [Online]. Available: https://tada.global

[40] T. Xu and Y. Cai, “Feeling-based location privacy protection for location-based services,” in *Proc. 16th ACM SIGSAC Conf. Comput. Commun. Secur.*, 2009, pp. 348–357.

[41] S. Holder, “Why uber drivers are fighting for their data,” 2019. [Online]. Available: https://www.bloomberg.com/news/articles/2019–08-22/why-uber-drivers-are-fighting-for-their-data

[42] K. Travers, “3 Questions: The price of privacy in ride-sharing app performance,” 2020. [Online]. Available: https://news.mit.edu/2020/3-questions-price-privacy-ride-sharing-app-performance-1014

[43] The European Parliament and the Council of the European Union, “Regulation (EU) 2016/679 of the european parliament and of the council of 27 April 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing Directive 95/46/EC (General Data Protection Regulation),” *Official J. Eur. Union*, L 119/1, 2016. [Online]. Available: https://op.europa.eu/en/publication-detail/-/publication/3e485e15-11bd-11e6-ba9a-01aa75ed71a1

[44] Y. Chen et al., “Demystifying hidden privacy settings in mobile apps,” in *Proc. IEEE 40th Symp. Secur. Privacy*, 2019, pp. 570–586.

[45] “Intel Xeon D-1702 processor,” 2022. [Online]. Available: https://ark.intel.com/content/www/us/en/ark/products/226115/intel-xeon-d1702-processor-5m-cache-up-to-1–70-ghz.html

[46] “The 5G intelligent Edge: Seamless Edge AI services for hyper-connected smart cities and transportation,” 2022. [Online]. Available: https://www.intel.com/content/dam/www/central-libraries/us/en/documents/capgemini-advantech-5g-rsu-solution-brief-2021–02-10.pdf

[47] “SOCKS protocol version 5,” 1996. Accessed: Jan. 20, 2022. [Online]. Available: https://datatracker.ietf.org/doc/html/rfc1928

[48] Y. Chen, M. Li, S. Zheng, D. Hu, C. Lal, and M. Conti, “One-time, oblivious, and unlinkable query processing over encrypted data on cloud,” in *Proc. 22nd Int. Conf. Informat. Commun. Secur.*, 2020, pp. 350–365.

[49] C. Garman, M. Green, and I. Miers, “Decentralized anonymous credentials,” in *Proc. 21st Netw. Distrib. System Secur. Symp.*, 2014, pp. 1–15.

[50] “Las Vegas Uber driver sexually assaults sleeping passenger, strangles woman as she begs him to stop: Reports,” 2021. Accessed: Jan. 22, 2022. [Online]. Available: https://www.foxnews.com/us/las-vegas-uber-driver-sexual-assault-passenger-strangle

[51] “Uber cab driver arrested for misbehaving with woman passenger,” 2019. Accessed: Jan. 22, 2022. [Online]. Available: https://www.thehindu.com/news/cities/bangalore/uber-cab-driver-arrested-for-misbehaving-with-woman-passenger/article29912810.ece

[52] “How star ratings work,” 2022. Accessed: Jan. 22, 2022. Available: https://www.uber.com/us/en/drive/basics/how-ratings-work

[53] S. Lee, M. Shih, P. Gera, T. Kim, H. Kim, and M. Peinado, “Inferring fine-grained control flow inside SGX enclaves with branch shadowing,” in *Proc. 26th USENIX Secur. Symp.*, 2017, pp. 557–574.

[54] S. Zhao, Q. Zhang, Y. Qin, W. Feng, and D. Feng, “SecTEE: A software-based approach to secure enclave architecture using tee,” in *Proc. 26th ACM SIGSAC Conf. Comput. Commun. Secur.*, 2019, pp. 1723–1740.

[55] G. Dessouky, T. Frassetto, and A.-R. Sadeghi, “Hybcache: Hybrid side-channel-resilient caches for trusted execution environments,” in *Proc. 29th USENIX Secur. Symp.*, 2020, pp. 451–468.

[56] I. Puddu, M. Schneider, M. Haller, and S. Capkun, “Frontal attack: Leaking control-flow in SGX via the CPU frontend,” in *Proc. 30th USENIX Secur. Symp.*, 2021, pp. 663–680.

[57] “Intel Celeron Processor G3900TE,” 2022. [Online]. Available: https://ark.intel.com/content/www/us/en/ark/products/90711/intel-celeron-processor-g3900te-2m-cache-2–30-ghz.html

[58] “Uber Announces Results for Third Quarter 2021,” 2022. [Online]. Available: https://investor.uber.com/news-events/news/press-release-details/2021/Uber-Announces-Results-for-Third-Quarter-2021

[59] M. Li, Y. Chen, C. Lal, and M. Conti, M. Alazab, and D. Hu, “Eunomia: Anonymous and secure vehicular digital forensics based on blockchain,” *IEEE Trans. Dependable Secure Comput.*, to be published, doi: 10.1109/TDSC.2021.3130583.

[60] M. Li, L. Zhu, Z. Zhang, C. Lal, M. Conti, and M. Alazab, “User-defined privacy-preserving traffic monitoring against n-by-1 jamming attack,” *IEEE/ACM Trans. Netw.*, to be published, doi: 10.1109/TNET.2022.3157654.

[61] Y. Lu, Y. Qi, S. Qi, Y. Li, H. Song, and Y. Liu, “Say no to price discrimination: Decentralized and automated incentives for price auditing in ride-hailing services,” *IEEE Trans. Mobile Comput.*, vol. 21, no. 2, pp. 663–680, Feb. 2022.

[62] D. Maram et al., “CanDID: Can-do decentralized identity with legacy compatibility, sybil-resistance, and accountability,” in *Proc. IEEE 42rd Symp. Secur. Privacy*, 2021, pp. 1348–1366.

[63] J. Ni, K. Zhang, Y. Yu, X. Lin, and X. Shen, “Privacy-preserving smart parking navigation supporting efficient driving guidance retrieval,” *IEEE Trans. Veh. Technol.*, vol. 67, no. 7, pp. 6504–6517, Jul. 2018.

[64] L. Wang, X. Lin, E. Zima, and C. Ma, “Towards Airbnb-like privacy-enhanced private parking spot sharing based on blockchain,” *IEEE Trans. Veh. Technol.*, vol. 69, no. 3, pp. 2411–2423, Mar. 2020.

[65] M. Li, L. Zhu, and X. Lin, “Privacy-preserving traffic monitoring with false report filtering via fog-assisted vehicular crowdsensing,” *IEEE Trans. Serv. Comput.*, vol. 14, no. 6, pp. 1902–1913, Nov./Dec. 2021, doi: 10.1109/TSC.2019.2903060.

[66] M. Li, J. Gao, Y. Chen, J. Zhao, and M. Alazab, “Privacy-preserving ride-hailing with verifiable order-linking in vehicular networks,” in *Proc. IEEE 19th Int. Conf. Trust Secur. Privacy Comput. Commun.*, 2020, pp. 599–606.

[67] S. T. Alshammari, A. Albeshri, and K. Alsubhi, “Building a trust model system to avoid cloud services reputation attacks,” *Egyptian Inform. J.*, vol. 22, pp. 493–503, 2021.

[68] A. Ugur, “Manipulator: A novel collusion attack on trust management systems in social IoT,” in *Proc. 10th Comput. Sci. On-Line Conf.*, 2021, pp. 578–592.

Meng Li (Member, IEEE) received the PhD degree in computer science and technology from the School of Computer Science and Technology, Beijing Institute of Technology, China, in 2019. He is an associate researcher and dean assistant with the School of Computer Science and Information Engineering, Hefei University of Technology, China. He is also a postdoctoral fellow with the Department of Mathematics and HIT Center, University of Padua, Italy, where he is with the Security and PRIvacy Through Zeal (SPRITZ) research group led by Prof. Mauro Conti. He was sponsored by ERCIM 'Alain Bensoussan' Fellowship Programme (from October 1, 2020 to March 31, 2021) to conduct postdoctoral research supervised by Prof. Fabio Martinelli at CNR, Italy. He was sponsored by China Scholarship Council (CSC) (from September 1, 2017 to August 31, 2018) for joint Ph.D. study supervised by Prof. Xiaodong Lin in the Broadband Communications Research (BBCR) Lab, University of Waterloo and Wilfrid Laurier University. His research interests include security, privacy, fairness, vehicular networks, applied cryptography, privacy computing, cloud computing, edge computing, and blockchain. In this area, he has published more than 40 papers in international peer-reviewed transactions, journals, magazines, and conferences, including *IEEE Transactions on Dependable and Secure Computing*, *IEEE/ACM Transactions on Networking*, *IEEE Transactions on Services Computing*, *IEEE Transactions on Network and Service Management*, *IEEE Transactions on Industrial Informatics*, *IEEE Transactions on Network Science and Engineering*, *IEEE Transactions on Green Communications and Networking*, *IoT Journal, Information Sciences*, *Future Generation Computer Systems*, *IEEE Communications Magazine*, *Wire versus Wireless Communication*, MobiCom, ICICS, SecureComm, TrustCom, and IPCCC.

Yifei Chen received the BE degree from the Hefei University of Technology, in 2019, and the MS degree from the School of Computer Science and Information Engineering, Hefei University of Technology. He was rewarded the China National Graduate Student Scholarship Award. He is now a security researcher with the Solution and Architecture Research Department, NSFOCUS. His research interests include cloud computing, blockchain, and privacy computing.

Chhagan Lal (Member, IEEE) received the PhD degree in computer science and engineering from the Malaviya National Institute of Technology, Jaipur, India, in 2014. He is currently working as a postdoctoral research fellow with the Department of Intelligent Systems, CyberSecurity Group, TU Delft, Netherlands. Previously, he was a postdoctoral research fellow with Simula Research Laboratory, Norway. He was a postdoctoral fellow with the Department of Mathematics, University of Padua, Italy, where he was part of the SPRITZ research group. During his PhD, he has been awarded with the Canadian Commonwealth Scholarship under the Canadian Commonwealth Scholarship Program to work in University of Saskatchewan, Saskatoon, SK, Canada. His current research interests include applications of blockchain technologies, security in software-defined networking, and Internet of Things networks.

Mauro Conti (Fellow, IEEE) received the PhD degree from the Sapienza University of Rome, Italy, in 2009. He is full professor with the University of Padua, Italy. He is also affiliated with TU Delft and University of Washington, Seattle. After his Ph.D., he was a postdoc researcher with Vrije Universiteit Amsterdam, The Netherlands. In 2011 he joined as Assistant Professor with the University of Padua, where he became associate professor in 2015, and full professor in 2018. He has been visiting researcher with GMU, UCLA, UCI, TU Darmstadt, UF, and FIU. He has been awarded with a Marie Curie Fellowship (2012) by the European Commission, and with a fellowship by the German DAAD (2013). His research is also funded by companies, including Cisco, Intel, and Huawei. His main research interest includes the area of security and privacy. In this area, he published more than 400 papers in topmost international peer-reviewed journals and conferences. He is editor-in-chief for *IEEE Transactions on Information Forensics and Security*, Area editor-in-chief for *IEEE Communications Surveys & Tutorials*, and has been Associate Editor for several journals, including *IEEE Communications Surveys & Tutorials*, *IEEE Transactions on Dependable and Secure Computing*, and *IEEE Transactions on Network and Service Management*. He was Program Chair for TRUST 2015, ICISS 2016, WiSec 2017, ACNS 2020, CANS 2021, and general chair for SecureComm 2012, SACMAT 2013, NSS 2021 and ACNS 2022. He is a senior member of the ACM, and fellow of the Young Academy of Europe.

Fabio Martinelli is a senior researcher with IIT-CNR, where he leads the cyber security project activities. He is co-author of about two hundreds scientific papers. His main research interests include involve security and privacy in distributed and mobile systems and foundations of security and trust. He is involved in several Steering Committees of international conferences, workshops, and working groups, like the IFIP WG 11.14 on secure services and software engineering. He manages R&D projects on information and communication security as FP6-FP7: Aniketos, Coco-Cloud, CAMINO, Connect, Contrail, SESAMO, Consequence, Sensoria, S3MS, GridTrust. Fabio Martinelli is also the coordinator of the EU FP7-NoE NESSoS on Engineering Secure Future Internet Services and He is also co-chair of the WG3 on Secure ICT research and innovation of the European Platform on Network and Information Security (NIS) a public/private cooperation promoted by the European Commission. He is also the project coordinator of the H2020 MSCA ITN NeCS (European Network in Cyber Security) as well as of the H2020 EU C3ISP project.

Mamoun Alazab (Senior Member, IEEE) received the PhD degree in computer science from the School of Science, Information Technology and Engineering, Federation University of Australia. He is an associate professor with the College of Engineering, IT and Environment with Charles Darwin University, Australia. He is a cybersecurity researcher and practitioner with industry and academic experience. His research is multidisciplinary that focuses on cybersecurity and digital forensics of computer systems with a focus on cybercrime detection and prevention including cyber terrorism and cyber warfare. He has more than 150 research papers. He delivered many invited and keynote speeches, 24 events in 2019 alone. He convened and chaired more than 50 conferences and workshops. He works closely with government and industry on many projects, including Northern Territory (NT) Department of Information and Corporate Services, IBM, Trend Micro, the Australian Federal Police (AFP), the Australian Communications and Media Authority (ACMA), Westpac, United Nations Office on Drugs and Crime (UNODC), and the Attorney General's Department. He is the founding chair of the IEEE Northern Territory (NT) Subsection.