QuickPool: Privacy-Preserving Ride-Sharing Service

Banashri Karmakar
IISc Bangalore
banashrik@iisc.ac.in

Shyam Murthy
IISc Bangalore
shyamsm1@iisc.ac.in

Arpita Patra
IISc Bangalore
arpita@iisc.ac.in

Protik Paul
IISc Bangalore
protikpaul@iisc.ac.in

***Abstract*—Online ride-sharing services (RSS) have become very popular owing to increased awareness of environmental concerns and as a response to increased traffic congestion. To request a ride, users submit their locations and route information for ride matching to a service provider (SP), leading to possible privacy concerns caused by leakage of users' location data. We propose QuickPool, an efficient SP-aided RSS solution that can obliviously match multiple riders and drivers simultaneously, without involving any other auxiliary server. End-users, namely, riders and drivers share their route information with SP as encryptions of the ordered set of points-of-interest (PoI) of their route from their start to end locations. SP performs a zone based oblivious matching of drivers and riders, based on partial route overlap as well as proximity of start and end points. QuickPool is in the semi-honest setting, and makes use of secure multi-party computation. We provide security proof of our protocol, perform extensive testing of our implementation and show that our protocol simultaneously matches multiple drivers and riders very efficiently. We compare the performance of QuickPool with state-of-the-art works and observe a speed up of 1.6 - 2×.**

I. Introduction

The European shared-mobility market is estimated to be about €70 billion as of 2022 [23] and projected to go to €150 billion by 2030. This projected rate of growth might not be surprising given the increasing awareness about environmental issues in the general population. Traditionally, ride-hailing service (RHS) refers to hailing a cab assisted by a central provider entity. Ride-sharing service (RSS) or carpooling, the focus of this work, is more eco-friendly than individually owned or RHS modes of transport, as sharing a ride means fewer cars on the road leading to lesser pollution, along with the advantages of dedicated high-occupancy-vehicle lanes. There are a number of Service Provider (SP) entities that provide RSS, namely QuickRide in India, BlaBlaCar, FlixCar in Europe, DiDi Chuxing in China, Waze, Scoop in the US, among many others. These SPs collect information from users both at subscription time as well as for individual rides, for billing and statistics purposes. SP would keep such information secure given the need to maintain its reputation, but there can be privacy-breach attacks [26] leading to identity theft and phishing attacks, or a malicious employee with access to information intending to obtain the same for personal gains. Such attacks might lead to privacy issues potentially resulting in financial penalties on SP [31]. Leakage of location information on riders and drivers might also lead to user profiling for stalking or other malicious activities. An article published in *TheJournal* reports of a gang in Dublin, Ireland, that made use of ride-hailing apps to target taxi drivers [32]. The article in TechCabal [30], an online media company in Nigeria reported many women getting stalked by drivers after multiple failed ride-hailing requests. In another incident reported from Istanbul, Turkey, drivers belonging to certain online ride apps were harassed and insulted by taxi drivers [15]. As is evident from these reports, individuals are vulnerable to harm when location and other private information are revealed. Therefore, there is a need to address the privacy concerns of both drivers and riders from each other as well as from SP.

RSS involves SP, and end-users, namely, riders and drivers. We note here that drivers are also riders but have a car and are willing to share their ride for benefits like sharing ride costs. While there are protocols that work in a decentralized model without involving an SP, some of which are described in Section I-A, in this work we consider a model that involves all three parties. This is reasonable from a practical standpoint, as we feel that in the RSS setting, users find it reassuring to be ride-sharing with a driver associated with a well-known service provider. It may be noted that even in the case of a decentralized model one could achieve comparable security, but there might be additional costs to realize the same. As noted earlier, a driver is another rider who is willing to share the ride with one or more "matching" rider(s). The criteria for matching are described in detail in the following sections. One of the primary functions of SP is to facilitate ride-matching. We consider SP's purview of operation (for e.g., a city and its suburbs) to be partitioned into suitable size grids such that the anonymity of parties inside the grid is well-preserved, similar to other works [34], [13]. SP keeps track of the grid ID of drivers when they advertise their availability for a ride (match request). Similarly, only when the rider sends a ride request, SP learns the grid ID. SP forwards the ride request to drivers in the rider's grid and compares the rider information with driver information in the same grid, for matching. No grid information is shared with SP by either party during the ride.

In privacy-preserving RSS, while the aforementioned matching procedure remains the same, the locations of all riders and drivers are hidden from each other and from SP during the ride-matching process. As described in Section II-B, we model SP as a semi-honest adversary that receives encrypted information from all end-user parties and follows the ride-matching protocol. In the process, it might try to obtain more information about the participating parties.

In RSS, two ride matching paradigms are considered in the literature [10], [1], [27]:

- Intersection based matching
- End-point based matching

At a high level, the intersection based matching protocol takes as inputs the routes of the rider and the driver, and returns a successful match if the length of the contiguous common path of the two parties is greater than a well-known threshold. The

term route will be explained in more detail in the following sections.

The end-point based matching protocol uses start and end points of the rider and driver to check for the proximity of the respective points to be within a well-known threshold distance to be deemed matchable. We use Euclidean distance (ED) as the metric. Given two points $X = (x_1, x_2)$ and $Y = (y_1, y_2)$, $\mathsf{ED}(X, Y) = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2}$. Both schemes are described in detail in the following sections.

In the privacy-preserving versions of the protocols, the encryptions of the routes as well as end-points are provided as inputs to the protocols. In this paper, we provide efficient implementations for privacy-preserving versions of both protocols. We do not intend to compare the features of one protocol versus the other. Both protocols are run simultaneously, and the choice of which one of the two protocol results to use is left to the rider.

In a secure privacy-preserving RSS (PP-RSS) solution, during ride establishment, the locations of riders and drivers are expected to be oblivious from each other as well as from SP. In other words, there should not be leakage of any location information to any of the drivers (or riders) about other riders (or drivers) except between the matched pairs of driver and rider, at the end of the protocol. We note that a system model that does not include an SP as a helper node, *SP-less model*, would inherently allow *all* matching drivers (or riders), that match itself, to be known to a given rider (driver). We refer to this as *match leakage*. Match leakages allow a rider to gain more information than it requires, such as how many drivers are traveling towards the direction of its choice. This information may potentially be used by a competitor company, acting as an adversary to launch location-harvesting attacks [20]. More leakages are possible when a set of riders or drivers collude with one another, but a secure protocol is expected to be secure against colluding groups of drivers, as well as against colluding groups of riders. In contrast to the SP-less model, in an SP-aided model, where the SP is modeled as a semi-honest adversary, the protocol should not leak any location information of drivers and riders to the SP as aforementioned. However, the SP is aware of all selected pairs of drivers and riders, which helps the system generate the maximum possible matching between riders and drivers. Moreover, a rider (driver) only learns with which driver (rider) it is matched. Thus, it manages to avoid the match leakage attack. In the literature, there are works [37], [14] that consider an auxiliary server along with SP, to realize efficient protocols. It is assumed that the auxiliary server does not collude with the SP, but in practice it maybe difficult to realize such non-colluding servers. Given that there might be a few hundred simultaneous ride requests received by the SP at some given point in time, it is imperative that ride-matching should be done without any perceptible delay. Thus, an efficient secure PP-RSS using lightweight symmetric-key primitives would be preferred over public-key operations. The riders and the drivers in a ride-sharing service are usually mobile low-end devices, therefore it is important for a practical solution that it should not have heavy computational requirements from the riders or the drivers. Additionally, protocols that make use of quantum-safe primitives would definitely be preferable to ones that are deemed quantum unsafe.

We ask the question whether we can have an efficient PP-RSS solution that works in an SP-aided model using only lightweight symmetric key quantum safe primitives, without using any auxiliary server, and provides only the matching specific information to only the selected driver-rider pair while maximizing the number of simultaneous matches. In this work, we answer that question in the affirmative.

A. Related works

We begin by listing a few works that operate in the decentralized model, for completeness, even though our model involves the SP. This is followed by works that either use an auxiliary server or make use of public-key primitives. Lastly, we list works that leak more information to riders than just the specific matched driver. Table I lists some of the related works and show one or more drawbacks in the earlier works.

	Resilient to match leakage	No auxiliary server	Quantum-safe/ symmetric-key
[10], [27]	✗	✓	✗
[2], [29]	✗	✓	✓
[37], [14], [21], [38]	✓	✗	✗
[1], [36]	✓	✓	✗
Our work	✓	✓	✓

TABLE I: Prior works listing one or more feature specific drawbacks of the protocol (need to fix the headings)

There are many works in the literature that deal with PP-RSS, starting with the work of Friginal et al. [8] where they give a dynamic carpooling architecture and identify various privacy assets that need to be preserved. One of the early PP-RSS protocols, PrivatePool, by Hallgren et al. [10] obliviously matches riders based on proximity matching of end points as well as overlaps in riders' route trajectories. Their work does not take into account the start time of travel and does not rely on a central SP for ride matching. TOPPool [27] is a follow up work by Pagnin et al. that incorporates the time of travel for ride matching. It made use of private set intersection for route segment matching and used an additive homomorphic encryption scheme along with a privacy-preserving matching method to aid in oblivious end point matching. The work of Raza et al. [2] is another decentralized ride-sharing scheme making use of public blockchain to ensure anonymity and accountability. Sanchez et al. [29] gave a decentralized RSS model that looked at peer trust enforcement using a distributed reputation management protocol.

The works PSRide by Yu et al. [37] and pShare by Huang et al. [14] made use of two non-colluding servers, namely the SP Server and a Crypto Server to securely evaluate Garbled Circuits for ride matching. The former work takes into account bounds on drop-off and pick-up times and the latter tries to reduce additional driver travel time for rider pick-up. The work by Li et al. [21] used road network embedding (RNE) to represent the road network and made use of additive homomorphic encryption (AHE) to securely compute distances between riders, and used an auxiliary non-colluding server for decryption. PGRide by Yu et al. [38] matched a group of riders

based on the smallest aggregate distance, aided by an external proxy server, using somewhat homomorphic encryption (SHE) to achieve matching. PRIS [11] by He et al. used map distance as the metric to match suitable riders aided by SP. Their method does an initial matching based on pre-selected spatial regions and then used locations encrypted using an AHE to filter in closeby riders.

Martelli et al. [22] compared location masking techniques like k-anonymity, l-diversity, and obfuscation, and showed that l-diversity using actual locations as points of interest (PoI) gives better performance. SRide [1] by Aïvodji et al. made use of secure multi-party computation (MPC) together with SHE to obliviously match riders and drivers. Their solution provided results for intersection based matching. In their solution, a rider gets to know all matched drivers which can lead to potential location-harvesting attacks by colluding riders, for example, [25]. A recent work called TAROT by Xu et al. [36] made use of Goldwasser-Micali (GM) public-key encryption scheme to match end-users using reverse minhash-based Jaccard similarity. Here also, the end-user decrypts all the encrypted matching results using her public-key, which can potentially lead to adversarial attacks.

B. Our contribution

As discussed in the prior section, to the best of our knowledge, most of the works either rely on heavy public-key primitives [29], [27], [10], [38] or make a strong assumption of the existence of additional auxiliary servers [14], [37], [38], [36], where such auxiliary servers perform key management and perform secure matching. This ensures privacy under the assumption that SP and the auxiliary servers do not collude. However, it is unclear how to realize such non-colluding servers in the context of RSS. As we have observed in Section I-A, [1], [36] may have potential privacy concerns due to leakage of additional information to riders, which we aim to circumvent by considering a different *system model*. Furthermore, the end-users of this application are equipped with low-end (limited computational resources) devices. Thus, the protocol should be extremely lightweight. Also, the system must be extremely fast and generate high throughput (number of matches in unit time) to facilitate its real-time nature. With the above goals in mind, we appropriately choose the system model and design the protocols to meet the above criteria. Below, we provide an overview and summarize the contributions of this work.

- *System model.* In this work, we consider PP-RSS in the centralized model, where the location privacy of end-users is preserved from the SP as well as other users. With the SP performing oblivious matching, we address the concerns discussed in the above paragraph, and our construction achieves the following properties. Here we emphasize that the privacy against an SP is maintained by modeling SP as a semi-honest adversary.

  Symmetric key versus public key primitives. Our construction relies only on symmetric key primitives and does not require any public key primitives. We rely only on pseudorandom generator (PRG), pseudorandom function (PRF), and secret sharing. As the symmetric key primitives are extremely lightweight compared to public key cryptography, our work achieves much better efficiency compared to prior works. Moreover, a lightweight construction is more suitable for low-end mobile devices, as the users (riders and drivers) are assumed to be mobile devices. Furthermore, since we do not need any public key cryptography, our construction does not rely on any specific hardness assumptions, thus our construction is plausibly post-quantum secure.

  Local maximum matching. In our construction, only SP obtains the output corresponding to every pair of riders and drivers whether they are matched. Therefore, SP can use these outputs to execute any local matching algorithm such that the maximum number of riders and drivers are matched. In prior works that consider a SP-less model, there does not exist any single party that gets the outcome of all the pairs, hence obtaining the maximum matching by executing any local algorithm is not possible.

  Strong security guarantee. In some of the prior works [1], [36], a rider learns about all drivers that satisfy the matching criteria with itself, thus weakening the security. In our construction, a rider (or driver) learns about only one driver (or rider) with whom it is matched.

- *Intersection based matching.* Using an approach similar to [27], we consider users with routes selected by their mobile apps. If a rider and a driver have significant overlap between their routes then they are matched. Executing the private set intersection (PSI) on the routes of the parties would give the desired result. [27] relies on oblivious pseudorandom function (OPRF), which is computationally expensive than standard PRF. Thus, we use PRF in place of OPRF. For this, we establish a common key between the parties. Using the common key, parties evaluate a PRF on the PoIs of their routes and send them to SP. Since the evaluated values are outputs of a PRF, they are random-looking. Hence, SP cannot infer any information about the received values. However, SP can locally find out the cardinality of the intersection of the involved parties. If the size of the intersection is greater than a predetermined threshold then the parties are considered as a match. We achieve significant speed-up compared to [27] e.g., [27] requires $O(n \log n)$ communication whereas we require $O(n)$, where n is the size of the route. We provide the details in Section III-A.

- *End-point based matching.* The second approach that we consider is end-point based matching. That is, if the starting and ending locations of the rider and the driver are 'nearby' then they are a match. We evaluate a function that outputs 1 if the Euclidean distance between start locations of driver and rider, and respectively between their end locations are lesser than a threshold. Observe that we evaluate the above-mentioned function where the comparands are not held by a single party. For this, we resort to techniques of multiparty computation (MPC) [3], [7], [28] where the inputs are secret shared among the SP, driver and rider as the three participating parties, and each party holds a share of the input. Parties perform the computation on their shares and maintain the sharing semantics. To complete the evaluation, parties are required to interact. We use MPC with 3 parties, secure against one semi-honest adversary, to compute the distances between the starting and ending locations of the rider and the driver. Finally, to compare the shared output of the above-mentioned computation with the publicly known

threshold value, we use function secret sharing [4], [16]. We provide the details in Section III-B.

• *Simultaneous matching.* Prior works have focused on checking if a single rider-driver pair satisfies the matching criteria. The users learn the outcome if the pair is a match or not. However, while processing multiple requests, it may be possible that some users get multiple matches whereas some others get very few matches. In such a case, if the matching for all the requests is not executed to maximize the number of matches that may lead to starvation of a set of users. We address this issue by matching multiple pairs together. For every pair of riders and drivers, SP learns if they are a match or not. With these outputs, SP executes a maximum bipartite matching to maximize the number of matches. We elaborate more on this in Section IV.

• *Benchmarks.* We implement both the end-point and intersection based matching algorithms, and compare the performance of the latter with the state-of-the-art protocol, O-PrivatePool from TOPPool [27]. We observe that QuickPool achieves almost $1.6-2\times$ speedup compared to O-PrivatePool. For the end-point based matching protocol, we can complete the execution of up to 40 riders and 40 drivers within 1 minute, whereas, for the intersection based matching protocol, within the same time, we can handle up to 100 riders and 100 drivers, considering the route length of the users to be at most 2048. Moreover, in the end-point matching protocol, each rider or driver needs to transmit significantly less data compared to SP. For instance, in scenarios with up to 100 riders and 100 drivers, each end-user needs to send only $\sim$3.13KB. QuickPool achieves a throughput of approximately 45.19 matches/min for end-point based matching and 323.79 matches/min for intersection based matching with a route length of 256, both in a scenario involving 60 riders and 60 drivers.

• *Scalability considerations.* In times of peak-traffic, it is quite likely that the number of drivers and riders can each be considerably high. In order to have a scalable solution, we propose to execute our matching in epochs of σ seconds, where σ is configurable by the SP based on real-time considerations like public events, peak-hour traffic etc. Additionally, SP will also be able to tune a parameter β, where β is the batch size of driver and rider requests on which the matching algorithms are executed. In other words, at the end of σ seconds, if in case we have more than β number of driver and rider unmatched requests, we pick the first β number of rider and driver requests, sorted on arrival time, and perform the match. Any unmatched requests in the current epoch get pooled together with incoming new requests and the matching process is repeated in the next epoch. We note here that executing the matching algorithm in batches might not give the global optimum maximum matching, but we adopt this method to be able to handle load surges gracefully. However, a resourceful SP with multiple servers can run multiple instances of β riders and drivers on separate servers in parallel, reducing the number of epochs and the overall execution time.

Organization. Section II introduces various primitives that we use throughout the work as well as the system and adversarial models that we consider in this work. Section III provides the technical details of our constructions for a pair of a rider and a driver. Matching multiple riders and drivers simultaneously is discussed in Section IV. Section V demonstrates the efficiency of our constructions by providing the benchmarking results. Finally, we conclude with a discussion in Section VI on obtaining a comprehensive privacy-preserving ride-sharing solution.

II. System Model and Preliminaries

A. Preliminaries

Points of interest (PoI) are geographical markers [22], like well-known landmarks, road intersection points, etc., that describe the route of a rider or driver. For start and end locations, the nearest PoI is considered. Details of each PoI are published by the SP (and available in the mobile app provided by the SP) along with its unique ID (which is represented as 32-bit non-negative integers) as well as its precise latitude and longitude coordinates. These (x, y) co-ordinates are represented in the UTM[1] format, where x and y are 32-bit non-negative integers, and represented in $\mathbb{Z}_{2^k}$ with a suitable value of k. We note here that PoIs give an additional location anonymity even after the ride is matched. However, during the ride matching process, only encryptions of PoI IDs (or their respective encrypted co-ordinates) are shared by users.

Route is an ordered set of PoIs of a rider or a driver, on the path from its start location to end location. A route is represented as a sequence of connected route segments between each PoI in the path. In other words, a route $\tau = \{(v_1, v_2), (v_2, v_3), \ldots, (v_{n-1}, v_n)\}$, where each v_i is a PoI identifier.

Private set intersection (PSI) is a cryptographic tool that allows two parties to compute the intersection of their respective sets without revealing any information about their individual sets to each other. [24] gives a literature survey of works on PSI. PSI has been used by many schemes in ride-sharing as well as ride-hailing settings [10], [27], [40], [39]. Prior works [10], [27] rely on this primitive to execute the intersection based matching protocols. Here, a rider and a driver execute the role of the two parties of PSI, and their respective routes are the input sets to the PSI protocol.

Secure multi-party computation (MPC) [3], [7] allows multiple distrusting parties to jointly and securely compute a common function of their inputs, without revealing the inputs or intermediate outputs to any of the other parties, and revealing only the final output to all parties. In order to accomplish this, each party creates shares of their input, in a way that the original data is revealed only when a specific number of shares (also known as threshold) are put together. Next, in the secret-sharing phase, each party receives a share from each of the other parties. Finally, they jointly proceed to compute the function with the help of their shares. The result is obtained by each party in secret-shared format which is reconstructed to obtain the result in plaintext format.

Function secret sharing (FSS) is a tool introduced by Boyle et al. [5] that facilitates evaluating a function f in a distributed manner. FSS splits the function into pieces such that the evaluations at any point on these pieces give an additive sharing of the function output. Among many functions, distributed point

[1] Universal Transverse Mercator coordinate system

function (DPF) and distributed comparison function (DCF) are popularly used in the setting where the function is split into two pieces. In this work, we use DCF to perform secure comparisons. Below we describe more on DCFs.

Distributed comparison function (DCF) [4] is a comparison function $f^{<}_{\alpha,\beta}$ outputting β if input $x > \alpha$ and zero otherwise. In the context of the current work, we consider a secure 2-party DCF (a rider and a driver) with a semi-honest dealer (SP).

B. System and adversarial model

We consider an online Ride-Sharing System (RSS) involving an SP. The SP receives encrypted information from each of the end parties, facilitates ride-matching, and informs the successful parties. The details of the matching algorithms are described in detail in Sections III-A and III-B.

We consider a semi-honest setting where all parties follow the protocol. We allow collusion among drivers and also collusion among riders. SP does not collude with drivers or riders, and drivers do not collude with riders. However, each party is interested to know more about the other parties than what is ordinarily available, and computable from the protocol transcription. In both the matching methods, the location information of each driver and rider is expected to be kept oblivious from SP and all other participating parties. After a successful match, only the two matched parties get to know each other's locations and none of the other participating drivers or riders learn anything.

C. Assumptions and limitations

In this work, we expect riders and drivers to make use of their mobile devices for ride and match requests via the mobile apps, provided by the SP at the time of subscription. The apps are responsible for keeping an updated view of the road network topology of the area of operations. Given the source and destination points of the end-users, the apps are expected to obtain an optimum route consisting of the closest PoIs, and follow the protocol as described in the paper. Any security breach from the point of view of the apps is out of the scope of this work. In our protocol, SP provides match results of both types of matching and the mobile apps should have the capability to convey the same to the user and solicit user input about which of the two results to use for final matching. Secure communication channels are assumed to be present between the end-users, and between SP and end-users.

III. QuickPool Single Driver-Rider Matching

As aforementioned, in RSS, the involved parties are a rider, R, and a driver D. A rider and a driver are termed 'matched' if certain criteria are met. These criteria ensure that R or D do not deviate from their intended route more than their acceptable deviation threshold. Considering such constraints, in the literature, primarily two paradigms are considered to decide if a R and D should be matched, namely, intersection based matching and end-point based matching. In this work, we consider both approaches.

In this section, we describe how a single pair of rider-driver matching is performed. In Section IV, we will see how to match more than one pair simultaneously. In summary, every rider-driver pair together with the SP executes the matching algorithms (Section III-A and Section III-B), and SP obtains the output for every possible pair. The algorithm in Section IV uses the output of Section III to execute a local matching algorithm to output multiple matching pairs. It then returns the matching result to each successful rider and driver. SP then restarts its matching process together with the unmatched riders and drivers, along with any newly available drivers' and riders' ride requests.

A. Intersection based matching

When a rider R raises a request for matching, it looks for a driver which has an aligning route with itself. Similarly, a driver D would prefer a rider with whom its route matches. This notion of preference is formalized in the form of intersection based matching, where R and D provide their routes, and if there is a significant overlap in the route (as specified by a threshold), they are matched. However, it is required that the routes of the users remain hidden. Below, we will first discuss how to check if a rider and a driver are matched in a privacy-preserving manner.

a) Matching a pair: Let R and D be a rider and a driver respectively with routes τ_{R} and τ_{D}. A route, τ, can be modeled as a path in a graph $G = (V, E)$ where G represents the road network of the operational area. V is the set of PoI identifiers, and $(u, v) \in E$ be an edge if u is 'reachable' from v. For simplicity, we assume that for a given source and destination pair, there is a unique designated route. Furthermore, a route is represented as a sequence of edges, that is $\tau = \{(v_1, v_2), (v_2, v_3), \ldots, (v_{n-1}, v_n)\}$. Along with the route, a user also provides the matching threshold, however, the threshold value does not need to be private. Consider R's inputs are the route τ_{R} and the threshold t_{R} and D's inputs are the route τ_{D} and the threshold t_{D}.

To check if R and D are a match we do the following:

$$f(\tau_{\mathsf{R}}, \mathsf{t}_{\mathsf{R}}, \tau_{\mathsf{D}}, \mathsf{t}_{\mathsf{D}}) = \begin{cases} 1 & \text{if } |\tau_{\mathsf{R}} \cap \tau_{\mathsf{D}}| \geq \max(\mathsf{t}_{\mathsf{R}}, \mathsf{t}_{\mathsf{D}}) \\ 0 & \text{if otherwise} \end{cases}$$

PrivatePool [10] provides a solution to the above problem by using threshold private set intersection (tPSI). However, the construction of tPSI has significant performance overhead (Sec. §1, pp. 2, [27]). TOPPool [27] addresses this issue by moving from tPSI to standard PSI without incurring any additional overhead. However, a PSI construction relies on public-key cryptographic primitives such as oblivious pseudo-random function (OPRF), which has significant computational overhead. In this work, we propose a solution without relying on any public-key cryptographic primitive, thus achieving an extremely fast and lightweight solution. We consider SP as one of the computational parties (without input) that facilitates the computation. On a very high level, our approach is the following. R and D establish a common key among themselves which is unknown to SP. They use the key to evaluate a PRF (not the costly OPRF) locally on the edges of their respective routes. They send the evaluated values to SP. SP obtains the cardinality of the intersection locally, if the threshold condition is met, it sets R and D as matched. Observe that the

values received by SP appear completely random as they are evaluations of PRF. We elaborate further on the construction below.

Let $\tau_{\mathsf{R}} = \{(v_1^{\mathsf{R}}, v_2^{\mathsf{R}}), \ldots, (v_{m-1}^{\mathsf{R}}, v_m^{\mathsf{R}})\}$ and $\tau_{\mathsf{D}} = \{(v_1^{\mathsf{D}}, v_2^{\mathsf{D}}), \ldots, (v_{n-1}^{\mathsf{D}}, v_n^{\mathsf{D}})\}$. The riders R and D establish a common key k by exchanging messages in a single round. R (D) samples a random value k_{R} (k_{D}) and sends it to D (R). R and D set $\mathsf{k} = \mathsf{k}_1 \oplus \mathsf{k}_2$ to be the common key. Let $\mathsf{F} : \{0,1\}^* \to \{0,1\}^*$ be a secure PRF. R and D evaluate the PRF, F, using the key k on the elements of their private inputs, that is,

$$\mathsf{R} \text{ computes } \mathsf{y}_j^{(\mathsf{R})} = \mathsf{F}(\mathsf{k}, v_j^{(\mathsf{R})} || v_{j+1}^{(\mathsf{R})}), \forall j \in [m-1],$$
$$\mathsf{D} \text{ computes } \mathsf{y}_j^{(\mathsf{D})} = \mathsf{F}(\mathsf{k}, v_j^{(\mathsf{D})} || v_{j+1}^{(\mathsf{D})}), \forall j \in [n-1].$$

R and D respectively send Y_{R} and Y_{D} to SP, where $Y_{\mathsf{R}} = \{\mathsf{y}_1^{(\mathsf{R})}, \ldots, \mathsf{y}_{m-1}^{(\mathsf{R})}\}$ and $Y_{\mathsf{D}} = \{\mathsf{y}_1^{(\mathsf{D})}, \ldots, \mathsf{y}_{n-1}^{(\mathsf{D})}\}$. Also, they send their corresponding threshold values, t_{R} and t_{D}. Note that, since the SP receives the PRF evaluations of the private inputs it does not learn any information about the underlying values. However, it can compute the cardinality of the intersection on the evaluated values. Furthermore, SP checks if the cardinality of the intersection is more than the provided threshold values, namely t_{R} and t_{D}. If the matching criteria are met, SP sets R and D as a match. Observe that, there are no messages exchanged between the rider and the driver that depend on their private inputs, hence privacy is guaranteed. Below, we describe the protocol Π_{Int}(Fig. 1) The protocol Π_{Int} requires 2 rounds of communication and $m+n+2$ elements. In this protocol, the computations on the riders' and drivers' sides are extremely lightweight (PRF evaluation on their local input), whereas the rest of the matching is performed by the service provider on PRF outputs.

Protocol Π_{Int}

Input:

- R's private input: $\tau_{\mathsf{R}} = \{(v_1^{(\mathsf{R})}, v_2^{(\mathsf{R})}), \ldots, (v_{m-1}^{(\mathsf{R})}, v_m^{(\mathsf{R})})\}$ and public input: t_{R}.
- D's private input: $\tau_{\mathsf{D}} = \{(v_1^{(\mathsf{D})}, v_2^{(\mathsf{D})}), \ldots, (v_{n-1}^{(\mathsf{D})}, v_n^{(\mathsf{D})})\}$ and public input: t_{D}.
- SP does not have any input in the protocol.

Output: 1 if $|\tau_{\mathsf{R}} \cap \tau_{\mathsf{D}}| \geq \max\{\mathsf{t}_{\mathsf{R}}, \mathsf{t}_{\mathsf{D}}\}$, otherwise 0

Protocol:

- R and D sample random values k_1 and k_2 locally and send it to D and R respectively.
- Both R and D locally set $\mathsf{k} = \mathsf{k}_1 \oplus \mathsf{k}_2$.
- R computes $\mathsf{y}_j^{(\mathsf{R})} = F(\mathsf{k}, v_j^{(\mathsf{R})} || v_{j+1}^{(\mathsf{R})})$ for all $j \in [m-1]$. Further, it sets $Y_{\mathsf{R}} = \{\mathsf{y}_1^{\mathsf{R}}, \ldots, \mathsf{y}_{m-1}^{\mathsf{R}}\}$.
- D computes $\mathsf{y}_j^{(\mathsf{D})} = F(\mathsf{k}, v_j^{(\mathsf{D})} || v_{j+1}^{(\mathsf{D})})$ for all $j \in [n-1]$. Further, it sets $Y_{\mathsf{D}} = \{\mathsf{y}_1^{\mathsf{D}}, \ldots, \mathsf{y}_{n-1}^{\mathsf{D}}\}$.
- R and D send $(Y_{\mathsf{R}}, \mathsf{t}_{\mathsf{R}})$ and $(Y_{\mathsf{D}}, \mathsf{t}_{\mathsf{D}})$ respectively to SP.
- SP computes $d = |Y_{\mathsf{R}} \cap Y_{\mathsf{D}}|$ and sets $\mathsf{Output} = 1$ if $d \geq \max\{\mathsf{t}_{\mathsf{R}}, \mathsf{t}_{\mathsf{D}}\}$ otherwise sets $\mathsf{Output} = 0$.

Fig. 1: Intersection based ride-sharing service

b) Security proof: Here, we will discuss the proof of security of our construction. We claim that out of the three parties (SP, R, D) if one of them is corrupt (semi-honest), the privacy of the honest parties will still be maintained. We establish our claim by showing that there is a simulator (probabilistic polynomial time algorithm) that generates an indistinguishable view of the corrupt parties without participating in the protocol. Due to the asymmetric nature of our protocol, we will first provide the simulator $\mathcal{S}_{SP}$ for a corrupt SP, and then we will argue why the simulated view is indistinguishable from the view of the corrupt party.

The simulator $\mathcal{S}_{SP}$ has the input b where $\mathsf{b} = 1$ if $|\tau_{\mathsf{R}} \cap \tau_{\mathsf{D}}| \geq \max\{\mathsf{t}_{\mathsf{R}}, \mathsf{t}_{\mathsf{D}}\}$, 0 otherwise. $\mathcal{S}_{SP}$ picks random routes τ'_{R} and τ'_{D} for R and D respectively such that $|\tau'_{\mathsf{R}} \cap \tau'_{\mathsf{D}}| = d$, where $d \geq \max\{\mathsf{t}_{\mathsf{R}}, \mathsf{t}_{\mathsf{D}}\}$ if $\mathsf{b} = 1$, otherwise $d < \max\{\mathsf{t}_{\mathsf{R}}, \mathsf{t}_{\mathsf{D}}\}$. It locally samples a random key k on behalf of R and D. Then, using τ'_{R} and τ'_{D}, it follows the protocol steps on behalf of R and D respectively. Note that SP receives $Y^{(\mathsf{R})}$ and $Y^{(\mathsf{D})}$ which are evaluations of PRF's, thus looks indistinguishable from the protocol messages.

Simulator $\mathcal{S}_{SP}$

Input Input to the simulator $\mathcal{S}_{SP}$ is $\mathsf{b} = 1$ if $|\tau_{\mathsf{R}} \cap \tau_{\mathsf{D}}| \geq \max\{\mathsf{t}_{\mathsf{R}}, \mathsf{t}_{\mathsf{D}}\}$, 0 otherwise.

- $\mathcal{S}_{SP}$ samples a random key k, integers m, n, d such that $m, n > d$ and $d \geq \max\{\mathsf{t}_{\mathsf{R}}, \mathsf{t}_{\mathsf{D}}\}$ if $\mathsf{b} = 1$, otherwise $d < \max\{\mathsf{t}_{\mathsf{R}}, \mathsf{t}_{\mathsf{D}}\}$.
- $\mathcal{S}_{SP}$ picks random routes for R and D in the following way.
  $\tau'_{\mathsf{R}} = \{(v_1, v_2), \ldots, (v_{d-1}, v_d), (v_d, v_{d+1}^{(\mathsf{R})}), \ldots, (v_{m-1}^{(\mathsf{R})}, v_m^{(\mathsf{R})})\}$
  $\tau'_{\mathsf{D}} = \{(v_1, v_2), \ldots, (v_{d-1}, v_d), (v_d, v_{d+1}^{(\mathsf{D})}), \ldots, (v_{n-1}^{(\mathsf{D})}, v_n^{(\mathsf{D})})\}$
- $\mathcal{S}_{SP}$ computes $\mathsf{y}_i = \mathsf{PRF}(\mathsf{k}, (v_i, v_{i+1}))$ for all $i \in [d-1]$, $\mathsf{y}_i^{(\mathsf{R})} = \mathsf{PRF}(\mathsf{k}, (v_i^{(\mathsf{R})}, v_{i+1}^{(\mathsf{R})}))$ for all $i \in [d, m-1]$ and $\mathsf{y}_i^{(\mathsf{D})} = \mathsf{PRF}(\mathsf{k}, (v_i^{(\mathsf{D})}, v_{i+1}^{(\mathsf{D})}))$ for all $i \in [d, n-1]$.
- $\mathcal{S}_{SP}$ sends $Y^{(\mathsf{R})} = \{\mathsf{y}_1, \ldots, \mathsf{y}_d, \mathsf{y}_{d+1}^{(\mathsf{R})}, \ldots, \mathsf{y}_m^{(\mathsf{R})}\}$ and $Y^{(\mathsf{D})} = \{\mathsf{y}_1, \ldots, \mathsf{y}_d, \mathsf{y}_{d+1}^{(\mathsf{D})}, \ldots, \mathsf{y}_n^{(\mathsf{D})}\}$ to SP, on behalf of R and D respectively.

Fig. 2: Simulator $\mathcal{S}_{SP}$ for Π_{Int} for a corrupt SP

Let $\mathcal{S}_{\mathsf{R}}$ be the simulator corresponding to a corrupt rider R. It interacts with R on behalf of SP and D. During the simulation, it samples a random key $[\mathsf{k}]_{\mathsf{D}}$ and sends it to R, on behalf of D. Then, on behalf of SP, it receives $Y^{(\mathsf{R})}$ as per protocol specification. Since D and SP do not send any messages to R, the simulated view is indistinguishable. Observe that the simulation for the corrupt driver is the same as that of the corrupt rider. Thus, we omit $\mathcal{S}_{\mathsf{D}}$.

Simulator $\mathcal{S}_{\mathsf{R}}$

Input Input to simulator $\mathcal{S}_{\mathsf{R}}$ is the route of the rider τ_{R}.

- $\mathcal{S}_{\mathsf{R}}$ samples a random key $[\mathsf{k}]_{\mathsf{D}}$,sends it to R on behalf of D.
- $\mathcal{S}_{\mathsf{R}}$ receives $Y^{(\mathsf{R})}$ as computed in Fig. 1.

Fig. 3: Simulator $\mathcal{S}_{\mathsf{R}}$ for Π_{Int} for a corrupt R

B. End-point based matching

Orthogonal to the intersection based matching, end-point based matching algorithm is well-studied in the literature. In this matching algorithm, if a driver and rider pair have the start position as well as the end positions close by, then they are a match. As aforementioned, to further preserve anonymity we use the PoI closest to the start or end location to identify the respective location, and refer to the UTM coordinates of the respective PoIs as the start and end locations. We compute the Euclidean distance between the rider and the driver, and if it is lesser than a predetermined threshold value then we consider them as a match. To compute the same, we rely upon MPC and FSS techniques.

a) Matching a pair: A rider R and a driver D are considered to be matched if they satisfy the following criteria.

1) The distance between the start locations of R and D is $\leq \rho_{\mathsf{s}}$, where ρ_{s} is the threshold start distance.
2) The distance between the end locations of R and D is $\leq \rho_{\mathsf{e}}$, where ρ_{e} is the threshold end distance.

In TOPPool [27], an additive homomorphic encryption scheme (Paillier scheme) is used to encrypt the locations of the rides, and a secure comparison algorithm is run to determine if any two riders can be matched by using an oblivious end-point matching algorithm. In our method, we securely compute the Euclidean distance, respectively, between the start locations, the end locations and use the respective threshold values to determine a successful match.

We perform the check in two steps. In the first step, R, D, and SP execute a 3-party MPC protocol, with 1 semi-honest corruption, to compute the square of the Euclidean distances of the corresponding start and end locations. This is followed by two secure comparison protocols.

3-party computation of Euclidean distance (ED*).* To check if R and D are a match or not, we compute the square of the Euclidean distance between the start locations of the R and D. Then we compare it with the square of the threshold value. We do the same for the end locations as well. If both conditions are met, then they are considered as a match. Below we describe how we compute the Euclidean distance between two secret locations (we describe the same for the start points. For the end points, it will follow similarly).

The riders and the drivers have their input as the start location and the end location. As aforementioned, a location is represented by its closest PoI and its UTM coordinates. Let R's inputs be $\mathsf{rLoc_s} = (\mathsf{rx_s}, \mathsf{ry_s})$ and $\mathsf{rLoc_e} = (\mathsf{rx_e}, \mathsf{ry_e})$, similarly, D's inputs be $\mathsf{dLoc_s} = (\mathsf{dx_s}, \mathsf{dy_s})$ and $\mathsf{dLoc_e} = (\mathsf{dx_e}, \mathsf{dy_e})$. We let $\mathsf{ED}(\mathsf{u}, \mathsf{v})$ to be the Euclidean distance between two points u and v. We need to obliviously compute $\mathsf{SDS} = [\mathsf{ED}(\mathsf{rLoc_s}, \mathsf{dLoc_s})]^2$ and $\mathsf{EDS} = [\mathsf{ED}(\mathsf{rLoc_e}, \mathsf{dLoc_e})]^2$.

Note that, for the above computation, one part of the input is held by R and the other part is available with D. We aim to compute the SDS and EDS without revealing anyone's input to any party other than the input holder. For this, we rely on the secure multiparty computation (MPC) tool, where an input holder splits the input into multiple parts in such a way that each of the parts does not contain any information about the input. In other words, each of the parts looks random and the input can be retrieved only if all the parts are put together. Splitting the input in such a way is commonly known as secret sharing. The computation of the desired function happens in two alternate stages, local computation and interaction. In the first step, R and D prepare for the secret sharing of their inputs. In the communication phase, they exchange the corresponding shares of the inputs. In the later phase, to compute the function, R and D perform local computation followed by another round of interaction to complete the distributed computation. Below we elaborate more on the local computation and communication.

Secret sharing semantics. In this work, we will use two types of secret sharing.

– *Additive secret sharing.* Let s be a secret (held by one of the two parties) and s is additively shared between R and D. $[\mathsf{s}]$ denotes the shares of s. $[\mathsf{s}]_{\mathsf{r}}$ and $[\mathsf{s}]_{\mathsf{d}}$ be the shares of R and D respectively. Note that, $[\mathsf{s}]_{\mathsf{d}} + [\mathsf{s}]_{\mathsf{r}} = \mathsf{s}$. Furthermore, $[\mathsf{s}]_{\mathsf{r}}$ and $[\mathsf{s}]_{\mathsf{d}}$ are random values, therefore individually these values do not reveal any information about s.

– *Augmented additive sharing.* As before, consider s to be a secret, and $[\![\mathsf{s}]\!]$ represents the augmented sharing of s. The augmented sharing of s consists of two components: a random mask (δ_{s}) and a masked value ($\mathsf{m_s} = \mathsf{s} + \delta_{\mathsf{s}}$). In the augmented sharing, both R and D hold the masked value $\mathsf{m_s}$; however, the mask (δ_{s}) is additively shared among the rider and the driver.

Note that, both the secret sharing mentioned above have the homomorphic property, that is, $[c_1\mathsf{x} + c_2\mathsf{y}] = c_1[\mathsf{x}] + c_2[\mathsf{y}]$ and $[\![c_1\mathsf{x} + c_2\mathsf{y}]\!] = c_1[\![\mathsf{x}]\!] + c_2[\![\mathsf{y}]\!]$ where c_1 and c_2 are publicly known values. Here the parties with their respective secrets perform the operations and obtain the shares of the computed values.

Key setup. For the above computation, R and D along with the service provider SP establish a set of common keys:

– R *and* D *obtain a common key* $\mathsf{k_{rd}}$. R samples a random key $[\mathsf{k_{rd}}]_1$ and sends it to D. Similarly, D samples a random key $[\mathsf{k_{rd}}]_2$ and sends it to R. Each party outputs $\mathsf{k_{rd}} = [\mathsf{k_{rd}}]_1 \oplus [\mathsf{k_{rd}}]_2$.

– R *and SP obtain a common key* $\mathsf{k_r}$. SP samples the key $\mathsf{k_r}$ and sends it to the rider R.

– D *and SP obtain a common key* $\mathsf{k_d}$. SP samples the key $\mathsf{k_d}$ and sends it to the driver D.

– R, D *and SP obtain a common key* $\mathsf{k_{all}}$. SP samples the key $\mathsf{k_{all}}$ and sends it to both the rider R and the driver D.

Sampling common random values. We will use the common keys to sample common random values. This will be required for the input sharing as well as for the computation. If more than one party is involved in sampling a random value, in that case, they will use the common key (as discussed in the above paragraph) to evaluate a PRF on an agreed value (e.g, a counter, ctr, which is incremented after each evaluation). For example, consider R and SP wish to sample a common random value, v, then both SP and R evaluate $\mathsf{PRF}(\mathsf{k_r}, \mathsf{ctr}) = \mathsf{v}$.

Input sharing. R has four input values $\mathsf{rx_s}$, $\mathsf{ry_s}$, $\mathsf{rx_e}$ and $\mathsf{ry_e}$. We aim to achieve augmented additive sharing of these values. To share these values, R and SP sample random values $\delta_{\mathsf{rx_s}}$, $\delta_{\mathsf{ry_s}}$, $\delta_{\mathsf{rx_e}}$ and $\delta_{\mathsf{ry_e}}$ using the key $\mathsf{k_r}$, these are the masks of the respective values. It is required to generate additive sharing of these

values. For that, SP and R sample $[\delta_{\mathsf{rx_s}}]_{\mathsf{r}}, [\delta_{\mathsf{ry_s}}]_{\mathsf{r}}, [\delta_{\mathsf{rx_e}}]_{\mathsf{r}}, [\delta_{\mathsf{ry_e}}]_{\mathsf{r}}$ using the key $\mathsf{k_r}$. SP computes $[\delta_{\mathsf{rx_s}}]_{\mathsf{d}} = \delta_{\mathsf{rx_s}} - [\delta_{\mathsf{rx_s}}]_{\mathsf{r}}$ and sends it to D. SP does the same for $[\delta_{\mathsf{ry_s}}]_{\mathsf{d}}, [\delta_{\mathsf{rx_e}}]_{\mathsf{d}}, [\delta_{\mathsf{ry_e}}]_{\mathsf{d}}$. Then, R locally computes $\mathsf{m_{rx_s}} = \mathsf{rx_s} + \delta_{\mathsf{rx_s}}$. Similarly, R computes $\mathsf{m_{ry_s}}, \mathsf{m_{rx_e}}$ and $\mathsf{m_{ry_e}}$. R sends $\mathsf{m_{rx_s}}, \mathsf{m_{ry_s}}, \mathsf{m_{rx_e}}$ and $\mathsf{m_{ry_e}}$ to D. To share D's input, D, R and SP follow the similar steps.

Computation of SDS and RDS by R and D. Using the homomorphic property of the $[\![\cdot]\!]$-sharing scheme, R and D compute $[\![\mathsf{rx_s}]\!] - [\![\mathsf{dx_s}]\!] = (\mathsf{m_{rx_s}} - \mathsf{m_{dx_s}}, [\delta_{\mathsf{rx_s}}] - [\delta_{\mathsf{dx_s}}])$. Similarly, R and D compute $[\![\mathsf{ry_s}]\!] - [\![\mathsf{dy_s}]\!]$, $[\![\mathsf{rx_e}]\!] - [\![\mathsf{dx_e}]\!]$ and $[\![\mathsf{ry_e}]\!] - [\![\mathsf{dy_e}]\!]$. Since

$$\mathsf{SDS} = \langle(\mathsf{rx_s} - \mathsf{dx_s}, \mathsf{ry_s} - \mathsf{dy_s}), (\mathsf{rx_s} - \mathsf{dx_s}, \mathsf{ry_s} - \mathsf{dy_s})\rangle$$
$$\mathsf{EDS} = \langle(\mathsf{rx_e} - \mathsf{dx_e}, \mathsf{ry_e} - \mathsf{dy_e}), (\mathsf{rx_e} - \mathsf{dx_e}, \mathsf{ry_e} - \mathsf{dy_e})\rangle,$$

we will describe how to compute the dot product $\mathsf{z} = \langle \mathbf{x}, \mathbf{y} \rangle$ where $\mathbf{x}$ and $\mathbf{y}$ are vectors of length 2 such that each component is an augmented additive share. Note that

$$[\![\mathbf{x}]\!] = ([\![\mathsf{x}_1]\!], [\![\mathsf{x}_2]\!]) = ((\mathsf{m}_{\mathsf{x}_1}, [\delta_{\mathsf{x}_1}]), (\mathsf{m}_{\mathsf{x}_2}, [\delta_{\mathsf{x}_2}])),$$
$$[\![\mathbf{y}]\!] = ([\![\mathsf{y}_1]\!], [\![\mathsf{y}_2]\!]) = ((\mathsf{m}_{\mathsf{y}_1}, [\delta_{\mathsf{y}_1}]), (\mathsf{m}_{\mathsf{y}_2}, [\delta_{\mathsf{y}_2}])).$$

Here $\delta_{\mathsf{x}_1}, \delta_{\mathsf{x}_2}, \delta_{\mathsf{y}_1}, \delta_{\mathsf{y}_2}$ are additively shared among R and D. However, SP holds these values in clear. To compute $\langle \mathbf{x}, \mathbf{y} \rangle$, SP prepares the following: $\delta_{\mathsf{xy}} = \delta_{\mathsf{x}_1} \cdot \delta_{\mathsf{y}_1} + \delta_{\mathsf{x}_2} \cdot \delta_{\mathsf{y}_2}$ and generates additive sharing of it. To generate the additive sharing of δ_{xy}, SP and R sample a common random value $[\delta_{\mathsf{xy}}]_{\mathsf{r}}$ using the common key $\mathsf{k_r}$. Then SP computes $[\delta_{\mathsf{xy}}]_{\mathsf{d}} = \delta_{\mathsf{xy}} - [\delta_{\mathsf{xy}}]_{\mathsf{r}}$ and sends it to D. Furthermore, SP, R and D prepare $[\cdot]$-sharing of a random value, say δ_{z}. For that SP and R sample a random value $[\delta_{\mathsf{z}}]_{\mathsf{r}}$ using their common key $\mathsf{k_r}$ and SP and D also sample another random value $[\delta_{\mathsf{z}}]_{\mathsf{d}}$ using their common key $\mathsf{k_d}$. SP sets $\delta_{\mathsf{z}} = [\delta_{\mathsf{z}}]_{\mathsf{r}} + [\delta_{\mathsf{z}}]_{\mathsf{d}}$. Observe that, R and D hold additive sharing of δ_{z}, the mask of $\langle \mathbf{x}, \mathbf{y} \rangle$. However, to maintain the sharing semantics, they are required to compute the masked value $\mathsf{m_z}$. Note that

$$\begin{aligned}
&\mathsf{z} = \mathsf{x}_1\mathsf{y}_1 + \mathsf{x}_2\mathsf{y}_2 \\
\Rightarrow\ &\mathsf{m_z} = (\mathsf{m}_{\mathsf{x}_1} - \delta_{\mathsf{x}_1})(\mathsf{m}_{\mathsf{y}_1} - \delta_{\mathsf{y}_1}) + (\mathsf{m}_{\mathsf{x}_2} - \delta_{\mathsf{x}_2})(\mathsf{m}_{\mathsf{y}_2} - \delta_{\mathsf{y}_2}) + \delta_{\mathsf{z}} \\
\Rightarrow\ &\mathsf{m_z} = \mathsf{m}_{\mathsf{x}_1}\mathsf{m}_{\mathsf{y}_1} + \mathsf{m}_{\mathsf{x}_2}\mathsf{m}_{\mathsf{y}_2} - \mathsf{m}_{\mathsf{x}_1}\delta_{\mathsf{y}_1} - \mathsf{m}_{\mathsf{y}_1}\delta_{\mathsf{x}_1} - \mathsf{m}_{\mathsf{x}_2}\delta_{\mathsf{y}_2} - \mathsf{m}_{\mathsf{y}_2}\delta_{\mathsf{x}_2} \\
&\quad + \delta_{\mathsf{x}_1}\delta_{\mathsf{y}_1} + \delta_{\mathsf{x}_2}\delta_{\mathsf{y}_2} + \delta_{\mathsf{z}}
\end{aligned}$$

Let $\delta_{\mathsf{xy}} = \delta_{\mathsf{x}_1}\delta_{\mathsf{y}_1} + \delta_{\mathsf{x}_2}\delta_{\mathsf{y}_2}$. R and D respectively compute

$$\begin{aligned}
[\mathsf{m_z}]_{\mathsf{r}} = \ & \mathsf{m}_{\mathsf{x}_1}\mathsf{m}_{\mathsf{y}_1} + \mathsf{m}_{\mathsf{x}_2}\mathsf{m}_{\mathsf{y}_2} - \mathsf{m}_{\mathsf{x}_1}[\delta_{\mathsf{y}_1}]_{\mathsf{r}} - \mathsf{m}_{\mathsf{y}_1}[\delta_{\mathsf{x}_1}]_{\mathsf{r}} \\
& - \mathsf{m}_{\mathsf{x}_2}[\delta_{\mathsf{y}_2}]_{\mathsf{r}} - \mathsf{m}_{\mathsf{y}_2}[\delta_{\mathsf{x}_2}]_{\mathsf{r}} + [\delta_{\mathsf{xy}}]_{\mathsf{r}} + [\delta_{\mathsf{z}}]_{\mathsf{r}} \qquad (1) \\
[\mathsf{m_z}]_{\mathsf{d}} = \ & -\mathsf{m}_{\mathsf{x}_1}[\delta_{\mathsf{y}_1}]_{\mathsf{d}} - \mathsf{m}_{\mathsf{y}_1}[\delta_{\mathsf{x}_1}]_{\mathsf{d}} \\
& - \mathsf{m}_{\mathsf{x}_2}[\delta_{\mathsf{y}_2}]_{\mathsf{d}} - \mathsf{m}_{\mathsf{y}_2}[\delta_{\mathsf{x}_2}]_{\mathsf{d}} + [\delta_{\mathsf{xy}}]_{\mathsf{d}} + [\delta_{\mathsf{z}}]_{\mathsf{d}}\,. \qquad (2)
\end{aligned}$$

To complete the computation, R and D exchange their shares of $\mathsf{m_z}$ with each other. R and D both locally compute $\mathsf{m_z} = [\mathsf{m_z}]_{\mathsf{r}} + [\mathsf{m_z}]_{\mathsf{d}}$. Therefore, $[\![\mathsf{z}]\!] = (\mathsf{m_z}, [\delta_{\mathsf{z}}])$.

Output. R, D and SP complete this phase of the computation by outputting $[\![\mathsf{SDS}]\!]$ and $[\![\mathsf{EDS}]\!]$ where SP holds δ_{SDS} and δ_{EDS} and R and D hold $\mathsf{m_{SDS}}$ and $\mathsf{m_{EDS}}$.

Performing the comparison: To decide if R and D are matched or not, it is required to check if $\mathsf{SDS} \le \rho_{\mathsf{s}}^2$ & $\mathsf{EDS} \le \rho_{\mathsf{e}}^2$ are satisfied. In the literature, there are different ways of performing computation. Considering bit representations of decimal values and the most significant bit (msb) as the sign bit, extracting the msb provides the output of the comparison with 0. This approach is widely used in various prior works [17], [6], [28], [12]. It requires evaluating a boolean circuit of depth $\log_2(k)$ (later works [19], [17], [28] optimized it evaluating a circuit with depth $\log_4(k)$), where the computation is performed over $\mathbb{Z}_{2^k}$. This approach incurs more interaction among the parties. Another line of works considers function secret sharing based approach [5], [4], [16], where to evaluate a comparison, a distributed comparison function (DCF) is evaluated, which is represented by a key. The keyholder locally evaluates the DCF using its key and obtains a random sharing of the output. However, generating the keys for the DCF is heavier when it is generated using MPC. Many prior works [16] bypass this challenge by considering an additional helper party other than the evaluators. Fortunately, in our case, the helper party is available naturally in the form of SP. In our construction, SP generates the DCF keys for R and D. Upon receiving the keys, R and D perform the DCF evaluation locally and obtain an additive sharing of the output. Below, we elaborate further on the execution of the comparison protocol using DCF. Consider a function $f_\alpha(\mathsf{x})$, parameterized by α, such that $f_\alpha(\mathsf{x}) = 1$ if $\mathsf{x} \le \alpha$, 0 otherwise.

The Gen algorithm of DCF takes α as an input and outputs K_0 and K_1. The helper party (SP) executes the Gen algorithm locally and sends K_0, K_1 to the evaluators. The evaluators (R and D) hold additive shares of the secret x and α. They reconstruct $\mathsf{x} + \alpha$ by exchanging the shares of $\mathsf{x} + \alpha$, which is obtained by locally applying the homomorphic property of the additive secret sharing. The evaluators perform the Eval algorithm locally to obtain the sharing of the comparison output. In this work, since SP holds δ_{SDS} and δ_{EDS} where these values are already additively shared with R and D, δ_{SDS} and δ_{EDS} are considered as the α of the corresponding comparison protocols. SP generates keys by providing δ_{SDS} and δ_{EDS} as input to the Gen algorithm. R and D upon receiving the keys perform the Eval on $\mathsf{m_{SDS}} - \rho_{\mathsf{s}}^2$ and $\mathsf{m_{EDS}} - \rho_{\mathsf{e}}^2$. R and D get the sharing of the output of the comparison protocol. They send the shares to SP, who reconstructs the output, and if both the conditions are met (that is, both the outputs are 1), it sets R and D as matched. We describe the Π_{EndPoint}(Fig. 4) protocol below. The communication cost of Π_{EndPoint} is 193 elements, and it requires 4 rounds of interaction.

Protocol $\Pi_{\mathsf{EndPoint}}(\mathsf{R}, \mathsf{D})$

Input:
- R's input $\mathsf{rx_s}$, $\mathsf{ry_s}$, $\mathsf{rx_e}$ and $\mathsf{ry_e}$.
- D's input $\mathsf{dx_s}$, $\mathsf{dy_s}$, $\mathsf{dx_e}$ and $\mathsf{dy_e}$.
- SP does not have any input in the protocol.
- Public input ρ_{s} and ρ_{e}.

Output: 1 if $\mathsf{SDS} \le \rho_{\mathsf{s}}^2$ & $\mathsf{EDS} \le \rho_{\mathsf{e}}^2$, 0 otherwise. Where
$\mathsf{SDS} = \langle(\mathsf{rx_s} - \mathsf{dx_s}, \mathsf{ry_s} - \mathsf{dy_s}), (\mathsf{rx_s} - \mathsf{dx_s}, \mathsf{ry_s} - \mathsf{dy_s})\rangle$
$\mathsf{EDS} = \langle(\mathsf{rx_e} - \mathsf{dx_e}, \mathsf{ry_e} - \mathsf{dy_e}), (\mathsf{rx_e} - \mathsf{dx_e}, \mathsf{ry_e} - \mathsf{dy_e})\rangle$

Protocol:
- *Input sharing.* For an input v, consider P (either R or D) to be the input owner.
  - SP and P sample a random mask δ_{v} and random shares δ_{v}, $[\delta_{\mathsf{v}}]_{\mathsf{R}}$. SP computes $[\delta_{\mathsf{v}}]_{\mathsf{D}} = \delta_{\mathsf{v}} - [\delta_{\mathsf{v}}]_{\mathsf{R}}$ and sends it to D.

– P computes $\mathsf{m_v} = \mathsf{v} + \delta_\mathsf{v}$ and sends it to the other party, that if $P = \mathsf{R}$, it sends it to the D and vice versa.

– R and D obtain $[\![\mathsf{v}]\!] = (\mathsf{m_v}, [\delta])$

• *Computation of* SDS *and* EDS.

– *Local computation.* Parties compute $[\![\mathsf{rx_s} - \mathsf{dx_s}]\!] = [\![\mathsf{rx_s}]\!] - [\![\mathsf{dx_s}]\!]$ and $[\![\mathsf{ry_s} - \mathsf{dy_s}]\!] = [\![\mathsf{ry_s}]\!] - [\![\mathsf{dy_s}]\!]$. Similarly, they compute $[\![\mathsf{rx_e} - \mathsf{dx_e}]\!]$ and $[\![\mathsf{ry_e} - \mathsf{dy_e}]\!]$.

– *Dot product.* Let $v_1 = \mathsf{rx_s} - \mathsf{dx_s}$, $v_2 = \mathsf{ry_s} - \mathsf{dy_s}$, $v_3 = \mathsf{rx_e} - \mathsf{dx_e}$, $v_4 = \mathsf{ry_e} - \mathsf{dy_e}$.

- SP computes $\delta_{12} = \delta_{v_1}\delta_{v_1} + \delta_{v_2}\delta_{v_2}$ and $\delta_{34} = \delta_{v_3}\delta_{v_3} + \delta_{v_4}\delta_{v_4}$. SP and R sample $[\delta_{12}]_\mathsf{R}$ and $[\delta_{34}]_\mathsf{R}$. Then, SP computes $[\delta_{12}]_\mathsf{D} = \delta_{12} - [\delta_{12}]_\mathsf{R}$ and $[\delta_{34}]_\mathsf{D} = \delta_{34} - [\delta_{34}]_\mathsf{R}$ and sends $[\delta_{12}]_\mathsf{D}$ and $[\delta_{34}]_\mathsf{D}$ to D. SP and R sample random values $[\delta_{\mathsf{SDS}}]_\mathsf{R}$ and $[\delta_{\mathsf{EDS}}]_\mathsf{R}$. Similarly, SP and D sample random values $[\delta_{\mathsf{SDS}}]_\mathsf{D}$ and $[\delta_{\mathsf{EDS}}]_\mathsf{D}$. SP sets $\delta_{\mathsf{SDS}} = [\delta_{\mathsf{SDS}}]_\mathsf{R} + [\delta_{\mathsf{SDS}}]_\mathsf{D}$ and $\delta_{\mathsf{EDS}} = [\delta_{\mathsf{EDS}}]_\mathsf{R} + [\delta_{\mathsf{EDS}}]_\mathsf{D}$.

- R and D computes $\mathsf{m_{SDS}}$ and $\mathsf{m_{EDS}}$ following (1) and (2) respectively.

• *Comparison.*

– SP execute DCF Gen algorithm with input δ_{SDS} and δ_{EDS} and generates the keys $K^\mathsf{r}_{\mathsf{SDS}}$, $K^\mathsf{d}_{\mathsf{SDS}}$, $K^\mathsf{r}_{\mathsf{EDS}}$ and $K^\mathsf{d}_{\mathsf{SDS}}$. SP sends $K^\mathsf{r}_{\mathsf{SDS}}$, $K^\mathsf{r}_{\mathsf{EDS}}$ to R and $K^\mathsf{d}_{\mathsf{SDS}}$, $K^\mathsf{d}_{\mathsf{EDS}}$ to D.

- R and D perform DCF Eval algorithm with $K^\mathsf{r}_{\mathsf{SDS}}$ and $K^\mathsf{r}_{\mathsf{EDS}}$ on input $\mathsf{m_{SDS}} - \rho_\mathsf{s}^2$ and $\mathsf{m_{EDS}} - \rho_\mathsf{e}^2$.

- R and D send their output to SP.

• *Output.* SP reconstructs the outputs. If both the outputs are 1, SP outputs 1, 0 otherwise.

Fig. 4: Endpoint based ride-sharing matching

b) Security proof: Here, we will discuss the security proof for the endpoint matching protocol. Similar to Section III-A0b, we will first discuss the simulator, $\mathcal{S}_{SP}$, for corrupt SP followed by the simulator, $\mathcal{S}_\mathsf{R}$, for corrupt R.

$\mathcal{S}_{SP}$ has inputs $\mathsf{b_{SDS}}$ and $\mathsf{b_{EDS}}$ where $\mathsf{b_{SDS}} = 1$ if $\mathsf{SDS} \leq \rho_\mathsf{s}^2$, 0 otherwise and $\mathsf{b_{EDS}} = 1$ if $\mathsf{EDS} \leq \rho_\mathsf{e}^2$, 0 otherwise. To simulate the view of SP, $\mathcal{S}_{SP}$ samples random inputs of R and D. With the random inputs, $\mathcal{S}_{SP}$, execute the steps of Π_{EndPoint} honestly on behalf of R and D. In the last step of the protocol, where R and D send their shares of the outputs, $\mathcal{S}_{SP}$ sends random additive sharing of $\mathsf{b_{SDS}}$ and $\mathsf{b_{EDS}}$ to SP. The simulated view is indistinguishable from the view of the corrupt SP of the protocol.

Simulator $\mathcal{S}_{SP}$

Input Inputs to simulator $\mathcal{S}_{SP}$ are the bits $\mathsf{b_{SDS}}$ and $\mathsf{b_{EDS}}$ where $\mathsf{b_{SDS}} = 1$ if $\mathsf{SDS} \leq \rho_\mathsf{s}^2$, 0 otherwise and $\mathsf{b_{EDS}} = 1$ if $\mathsf{EDS} \leq \rho_\mathsf{e}^2$, 0 otherwise respectively.

• $\mathcal{S}_{SP}$ samples random inputs on behalf of R and D and execute the protocol steps until the last interaction from R, D to SP, honestly on behalf of R and D.

• In the last step, $\mathcal{S}_{SP}$ sends an additive sharing of $\mathsf{b_{SDS}}$ and $\mathsf{b_{EDS}}$ to SP on behalf of R and D.

Fig. 5: Simulator $\mathcal{S}_{SP}$ for Π_{EndPoint} for a corrupt SP

$\mathcal{S}_\mathsf{R}$ has the inputs of the corrupt rider R, that is, $(\mathsf{rx_s}, \mathsf{ry_s})$, $(\mathsf{rx_e}, \mathsf{ry_e})$. $\mathcal{S}_\mathsf{R}$ needs to simulate the view of R. The view of R comprises the exchanged messages during the input sharing, dot product, and DCF evaluations. In the input sharing phase, corresponding to R's input $\mathcal{S}_\mathsf{R}$ (on behalf of SP) and R, sample the masks for the inputs and the shares of the masks. Then R sends the masked values to D, $\mathcal{S}_\mathsf{R}$ receives the masked values on behalf of D. Corresponding to D's inputs, on behalf of SP, $\mathcal{S}_\mathsf{R}$ and R sample shares of the random masks. Here note that D's inputs are not known to $\mathcal{S}_\mathsf{R}$. However, it needs to prepare the masked values on behalf of D and send them to R. $\mathcal{S}_\mathsf{R}$ samples random values as the masked values and sends them to D. Since the masks are unknown to R, the masked values are indistinguishable from randomly sampled values. Thus, this step of the simulation is indistinguishable. To perform the dot product, SP generates correlated randomness δ_{12}, δ_{13}, δ_{SDS} and δ_{EDS}. These values are additively shared between R and D. Therefore, on behalf of SP, $\mathcal{S}_\mathsf{R}$ along with R sample random values $[\delta_{12}]_\mathsf{R}$, $[\delta_{34}]_\mathsf{R}$, $[\delta_{\mathsf{SDS}}]_\mathsf{R}$ and $[\delta_{\mathsf{EDS}}]_\mathsf{R}$. To complete the computation of the dot product, R and D need the masked values, $\mathsf{m_{SDS}}$ and $\mathsf{m_{EDS}}$. $\mathcal{S}_{SP}$ sends uniformly sampled $[\mathsf{m_{SDS}}]_\mathsf{D}$ and $[\mathsf{m_{EDS}}]_\mathsf{D}$ to R and it receives $[\mathsf{m_{SDS}}]_\mathsf{R}$ and $[\mathsf{m_{EDS}}]_\mathsf{R}$ on behalf of D. All these messages in the protocol are also uniformly distributed; hence, the simulated view is indistinguishable from the view of R. Finally, for the DCF evaluations, the keys $K^\mathsf{r}_{\mathsf{SDS}}$ and $K^\mathsf{r}_{\mathsf{EDS}}$ are uniformly distributed. Therefore, $\mathcal{S}_\mathsf{R}$ sends uniformly sampled keys, $K^\mathsf{r}_{\mathsf{SDS}}$ and $K^\mathsf{r}_{\mathsf{EDS}}$ to R on behalf of SP and it receives the shares of the outputs of the comparisons from R. The simulated view is indistinguishable from the view of R. The simulator for the corrupt driver, $\mathcal{S}_\mathsf{D}$, is similar to $\mathcal{S}_\mathsf{R}$.

Simulator $\mathcal{S}_\mathsf{R}$

Input Input to simulator $\mathcal{S}_\mathsf{R}$ is the starting and ending points of R, that is, $(\mathsf{rx_s}, \mathsf{ry_s})$, $(\mathsf{rx_e}, \mathsf{ry_e})$.

• **Input sharing.**

- R*'s input.*

• $\mathcal{S}_\mathsf{R}$ and R sample δ_v for all $\mathsf{v} \in \{\mathsf{rx_s}, \mathsf{ry_s}, \mathsf{rx_e}, \mathsf{ry_e}\}$.

• $\mathcal{S}_\mathsf{R}$ and R sample $[\delta_\mathsf{v}]_\mathsf{R}$ for all $\mathsf{v} \in \{\mathsf{rx_s}, \mathsf{ry_s}, \mathsf{rx_e}, \mathsf{ry_e}\}$.

• $\mathcal{S}_\mathsf{R}$ receives $\mathsf{m_v}$ on behalf of D, for all $\mathsf{v} \in \{\mathsf{rx_s}, \mathsf{ry_s}, \mathsf{rx_e}, \mathsf{ry_e}\}$.

- D*'s input.*

• $\mathcal{S}_\mathsf{R}$ and R sample random $[\delta_{\mathsf{dx_s}}]_\mathsf{R}$, $[\delta_{\mathsf{dy_s}}]_\mathsf{R}$, $[\delta_{\mathsf{dx_e}}]_\mathsf{R}$, $[\delta_{\mathsf{dy_e}}]_\mathsf{R}$.

• $\mathcal{S}_\mathsf{R}$ samples $\mathsf{m_{dx_s}}$, $\mathsf{m_{dy_s}}$, $\mathsf{m_{dx_e}}$, $\mathsf{m_{dy_e}}$ and sends them to R.

• **Local computation.** To simulate the local computations, $\mathcal{S}_\mathsf{R}$ follows the protocol steps on behalf of D and SP.

• **Dot Product.**

- *Simulating the masks generation.* $\mathcal{S}_\mathsf{R}$ along with R sample $[\delta_{12}]_\mathsf{R}$, $[\delta_{34}]_\mathsf{R}$, $[\delta_{\mathsf{SDS}}]_\mathsf{R}$ and $[\delta_{\mathsf{EDS}}]_\mathsf{R}$ (described in Fig. 4) on behalf of SP.

- *Simulating the masked value.* $\mathcal{S}_\mathsf{R}$ samples random $[\mathsf{m_{SDS}}]_\mathsf{D}$, $[\mathsf{m_{EDS}}]_\mathsf{D}$ and sends it to R on behalf of D and it receives $[\mathsf{m_{SDS}}]_\mathsf{R}$, $[\mathsf{m_{EDS}}]_\mathsf{R}$ from R on behalf of D.

• **DCF evaluation.** $\mathcal{S}_\mathsf{R}$ random keys $K^\mathsf{r}_{\mathsf{SDS}}$ and $K^\mathsf{r}_{\mathsf{EDS}}$ and sends them to R on behalf of SP. It receives shares of outputs of the comparisons from R on behalf of SP.

Fig. 6: Simulator $\mathcal{S}_\mathsf{R}$ for Π_{EndPoint} for a corrupt R

IV. Matching multiple pairs

In Section III-A and Section III-B, we describe two different procedures to check if a rider and a driver are a match. Let $\Pi(\mathsf{R}, \mathsf{D})$ indicates if R and D are a match or not, where Π can be instantiated using Π_{Int} or Π_{EndPoint}. SP obtains the output, $\Pi(\mathsf{R}, \mathsf{D})$ for every pair R and D. Note that it is possible that some riders may be matched with a common driver, where one of the riders may have a match with another driver, but the other rider does not have any other matches. In that case, random assignments of rider-driver pairs may lead to starvation of some users (remain unmatched). We elaborate on this with an example below. Consider the following scenario with 2 riders, R_1, R_2 and 2 drivers, D_1, D_2 such that $\Pi(\mathsf{R}_1, \mathsf{D}_1) = 1$, $\Pi(\mathsf{R}_1, \mathsf{D}_2) = 1$, $\Pi(\mathsf{R}_2, \mathsf{D}_1) = 1$ and $\Pi(\mathsf{R}_2, \mathsf{D}_2) = 0$. In the above example, if R_1 and D_1 are matched, then both R_2 and D_2 will remain unmatched. However, $(\mathsf{R}_1, \mathsf{D}_2)$ and $(\mathsf{R}_2, \mathsf{D}_1)$ is an optimal match. Given a set of riders and drivers, it is possible to find the optimal matching by executing a maximum bipartite matching. Since SP gets the output of the matching protocol for every pair in clear, it can construct the adjacency matrix of the corresponding bipartite graph and execute the matching algorithm locally. Therefore, any state-of-the-art maximum matching algorithm can be used to obtain the optimum matching without incurring any communication overhead. Furthermore, the computation overhead will be towards SP only, depending on the underlying matching algorithm. Also, in the case of the intersection based matching algorithm, it is easy to obtain a preference list (based on the cardinality of the intersection set) corresponding to every rider and driver. For a rider R_i, let $\mathcal{D}^{(i)} = \{\mathsf{D}_1^{(i)}, \ldots, \mathsf{D}_{m_i}^{(i)}\}$ be the preference list. That is, $\forall j \in [m_i]$, $f(\tau_{\mathsf{R}_i}, \mathsf{t}_{\mathsf{R}_i}, \tau_{\mathsf{D}_j}, \mathsf{t}_{\mathsf{D}_j}) = 1$ and $|\tau_{\mathsf{R}_i} \cap \tau_{\mathsf{D}_j}| \geq |\tau_{\mathsf{R}_i} \cap \tau_{\mathsf{D}_{j+1}}|$ for all $j \in [m_i - 1]$. Similarly, driver D_j has the preference list $\mathcal{R}^{(j)} = \{\mathsf{R}_1^{(j)}, \ldots, \mathsf{R}_{n_j}^{(j)}\}$. Then, SP can also locally execute the stable matching algorithm [9].

When a rider (driver) raises a ride (match) request, it is checked with all the available drivers (riders) to see if they are a match. As discussed above, we execute a bipartite maximum matching. For that, we consider a bipartite graph, $G = (V, E)$ where $V = \mathcal{R} \cup \mathcal{D}$, $\mathcal{R}$ and $\mathcal{D}$ are the set of all riders and drivers, respectively. Note that, $\mathcal{R}$ and $\mathcal{D}$ form a partition of V. We consider $(\mathsf{R}, \mathsf{D}) \in E$ to be an edge in the graph if $\mathsf{R} \in \mathcal{R}$ and $\mathsf{D} \in \mathcal{D}$ are matched, that is, $\Pi_{\mathsf{EndPoint}}(\mathsf{R}, \mathsf{D}) = 1$. Thus, we execute a maximum bipartite matching algorithm on G to obtain the maximum number of matches. It is easy to see that a maximum matching in G will give a maximum possible matching between the set of riders and drivers.

V. Experiments and Results

In this section, we evaluate the effectiveness of QuickPool in providing an efficient ride-sharing solution. Firstly, we describe the implementation details of QuickPool and then we benchmark our proposed intersection based matching and end-point based matching protocols and show that compared to previous works [27], our protocols achieve significant speed-up without compromising user privacy.

A. Implementation details

We now outline the implementation details of QuickPool, which we utilize for benchmarking the intersection based matching and end-point based matching protocols.

a) Environment: The benchmarks are conducted over MAN network simulated on a local server, equipped with a 4.5GHz, 24 cores, 48 threads, AMD Ryzen™ Threadripper PRO 5965WX processor having 256GB RAM. Each rider and driver as well as the SP are executed as multiple processes on a single server. We consider a bandwidth of 500 Mbps and 10 ms latency and use the Linux `tc` command from network emulation package `netem` to emulate a MAN network for inter-process communication. Both intersection based matching and end-point based matching protocols are benchmarked in MAN, as QuickPool, being a ride-sharing solution, is typically more suited for MAN environments. Each of the test suites is executed ten times without interruption and the average run time is reported in this paper.

b) Implementation of QuickPool: We implement QuickPool in C++17, leveraging the codebase of Asterisk [17] and EMP toolkit [35][2]. We set the computational security parameter $\kappa = 128$ and ensure a statistical security of at least 2^{-32}. To implement the intersection based matching protocol, we utilize the AES implementation from the EMP toolkit, which internally utilizes highly efficient AES-NI. We utilize funshade [16] implementation for DCF instantiations which are used for performing secure comparisons.

c) Instantiation of other Protocols: We compare our intersection based matching protocol with the O-PrivatePool protocol introduced in TOPPool [27], which internally employs the BaRK-OPRF PSI protocol [18]. The BaRK-OPRF PSI protocol is recognized as one of the fastest state-of-the-art PSI protocols for large batches of data. Since the implementation of O-PrivatePool is not publicly available, we base our comparison on the BaRK-OPRF PSI protocol (utilizing the open-source code accessible at [33]), which we run on our server in MAN settings. Given that O-PrivatePool relies on BaRK-OPRF, we assert that O-PrivatePool will take at least as much time as BaRK-OPRF. We are unable to compare our end-point based matching algorithm with prior works, as to the best of our knowledge, there is no publicly available implementation accessible for such algorithms.

B. Performance benchmarks of QuickPool

We now benchmark the intersection-based and end-point-based matching protocols, highlighting their efficiency in terms of both communication cost and runtime.

a) Intersection based matching: We analyze the performance of the intersection based matching protocol (Fig. 1) by setting the number of riders equal to the number of drivers and varying both from 1 to 100, as depicted in Fig. 7. Additionally, we also vary the set size from 32 up to 4096, where by 'set size' we mean the route length of each user. We observe that even with a set size of up to 2048, considering the number of riders and drivers to be up to 100, per party communication cost remains almost within 3MB. Note that in this protocol, SP does not send any data to any other party throughout the process. Regarding runtime, with a set size of up to 2048 and up to 100 riders and drivers, the total runtime is below 1 minute. Additionally, we report the throughput for set a size of

[2]Our implementation is publicly available at: https://github.com/cris-coders-iisc/QuickPool. Our code is developed for benchmarking and is not optimized for industry-grade use.

256 in Table II, which captures the number of rider-driver pairs matched per minute and is computed as throughput (/minute) = $\min(\#\text{ riders}, \#\text{ drivers}) * 60/\text{run time}$.

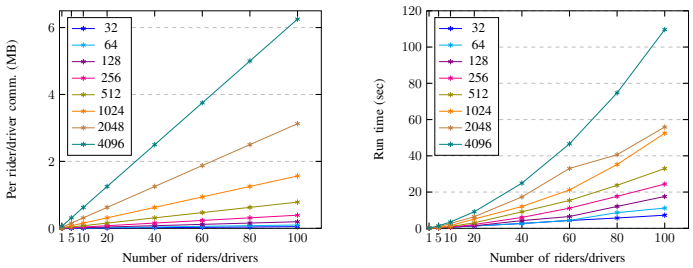

Fig. 7: Performance of intersection based matching algorithm over a MAN network, for varying numbers of riders/drivers in $\{\mathbf{1}, \mathbf{5}, \mathbf{10}, \mathbf{20}, \mathbf{40}, \mathbf{60}, \mathbf{80}, \mathbf{100}\}$ and set size in $\{\mathbf{2^j}\}_{\mathbf{j}\in[\mathbf{5},\mathbf{12}]}$.

# riders/ drivers	Intersection based		End-point based	
	Run time (sec)	Throughput (/min)	Run time (sec)	Throughput (/min)
5	0.31426	954.62	0.94	319.15
10	0.62484	960.25	2.88	208.33
20	2.10365	570.44	9.88	121.46
40	6.01544	398.97	36.45	65.84
60	11.11847	323.79	79.67	45.19
80	17.64009	272.11	139.65	34.37
100	24.41076	245.79	216.86	27.67

TABLE II: Run time and throughput (per minute) for intersection (considering set size = 256) and end-point based matching algorithms for varying numbers of riders/drivers.

b) End-point based matching: Similar to the intersection based matching protocol, we analyze the performance of the end-point based matching protocol (Fig. 4) by varying the number of riders and drivers, as depicted in Fig. 8. We observed that even with 100 riders and drivers, the total communication remains below 30MB, while with up to 40 riders and drivers, the communication cost is less than 5MB. Notably, in this protocol, each rider and driver transmits significantly very less data compared to SP. For instance, in a scenario with 100 riders and drivers, each of the riders and drivers needs to send only ∼3.13KB, as shown in Table III. Note that the amount of data sent by SP is comparatively higher, as it needs to transmit the DCF keys to both the rider and the driver. Regarding runtime, this protocol completes its execution very quickly. For instance, with up to 40 riders and drivers, it takes less than 1 minute to complete. Additionally, we report the throughput in Table II for varying number of riders and drivers. Note that although the throughput decreases with the increase of users due to increased computation and runtime, the likelihood of successful matching in real scenarios increases. This is because there are more matching options available for each rider/driver.

C. Comparison with previous works

The BaRK-OPRF protocol operates between a single pair of parties for computing private set intersection. We conducted

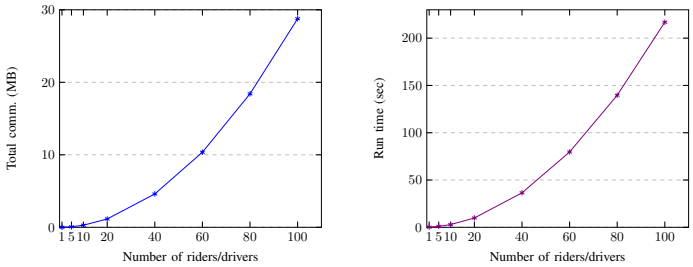

Fig. 8: Performance of end-point based matching algorithm over a MAN network, for varying numbers of riders/drivers in $\{\mathbf{1}, \mathbf{5}, \mathbf{10}, \mathbf{20}, \mathbf{40}, \mathbf{60}, \mathbf{80}, \mathbf{100}\}$.

# riders /drivers	1	5	10	20	40	60	80	100
sent by SP	2.91	72.19	288.52	1153.59	4613.44	10379.53	18451.88	28830.47
sent by each user	0.031	0.16	0.31	0.63	1.25	1.88	2.5	3.13

TABLE III: Communication cost (in KB) of SP, riders and drivers in end-point based matching over MAN for varying numbers of users (riders/drivers)

experiments with BaRK-OPRF, varying the set size from 32 to 4096, and compared both of its per party communication cost and runtime with our intersection-based matching protocol for a single rider and driver. Our observations indicate that our protocol significantly outperforms BaRK-OPRF. For instance, even with a set size of 4096, our protocol's execution time is ∼106ms, while BaRK-OPRF takes longer, with ∼140ms even for a set size of 32. Fig. 9 illustrates this comparison for other set sizes as well. We observe that QuickPool achieves 1.6 - 2× and at least 8× improvement over BaRK-OPRF in terms of run time and communication cost respectively.

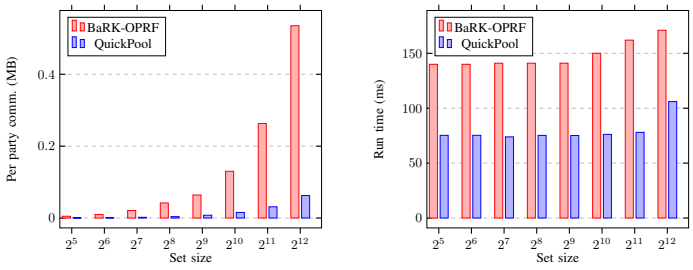

Fig. 9: Performance comparison over a MAN network between BaRK-OPRF and QuickPool for a single rider and driver, for varying set size in $\{\mathbf{2^j}\}_{\mathbf{j}\in[\mathbf{5},\mathbf{12}]}$.

VI. Towards a Comprehensive and Practical Solution

In this work, we propose algorithms for both intersection based and end-point based driver-rider matching that are more efficient compared to prior works. We also demonstrate their effectiveness for multiple simultaneous matching. Our works can be used as building blocks that an SP can use to build secure applications. However, a few more steps are needed to make our algorithms into a practical solution. We will briefly discuss them below.

Scalability. Our matching algorithms will be executed every epoch of σ duration. SP can set σ, and β, the number of unmatched rider and driver requests, by using results in Table II as a reference. However, the SP can vary these parameters based on incoming ride request variations due to time-of-day, weather-based traffic conditions and other practical considerations.

Time matching. TOPPool [27] proposes matching a rider and a driver, not just with either the intersection or end-point matching, but also their respective ride times. To facilitate the time-based matching in QuickPool, we need to perform one additional DCF to compare the differences between the rider's and the driver's ride times with a publicly known end-user-specific threshold value. The result of the comparison can be used as a match criteria together with either intersection based or end-point based matching.

Group matching. In group matching [38], multiple riders who satisfy the matching criteria (significant overlap in the respective routes or nearby start and end-points) are considered as a single group. Then, a representative of a group is matched with a driver. One can trivially extend the rider-driver matching protocol in QuickPool to perform group matching. First, we compare pairs of riders to obtain a matched group. Next, a representative of this matched group can execute a matching protocol with the drivers. Here, the matching protocol may not necessarily be ride-sharing matching. One can use ride-hailing matching as well where the driver's location should be close to the riders' locations. For this, we can use a relaxed version of end-point based matching where we check if the difference between the start location of the driver and the start location of the rider is less than a threshold value.

Other desirable features. A secure and anonymous payment system such as eCash or equivalent needs to be integrated into the system for fare payment after ride completion. Similar to features available in existing ride matching services, an anonymous rating system to mutually rate riders and drivers can be implemented for quality improvement and reputation maintenance.

REFERENCES

[1] U. M. Aïvodji, K. Huguenin, M.-J. Huguet, and M.-O. Killijian, "Sride: A privacy-preserving ridesharing system," in *Proceedings of the 11th ACM Conference on Security & Privacy in Wireless and Mobile Networks*, 2018.

[2] M. Baza, N. Lasla, M. M. Mahmoud, G. Srivastava, and M. Abdallah, "B-ride: Ride sharing with privacy-preservation, trust and fair payment atop public blockchain," *IEEE Transactions on Network Science and Engineering*, 2019.

[3] M. Ben-Or, S. Goldwasser, and A. Wigderson, "Completeness theorems for non-cryptographic fault-tolerant distributed computation," in *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing*, 1988.

[4] E. Boyle, N. Chandran, N. Gilboa, D. Gupta, Y. Ishai, N. Kumar, and M. Rathee, "Function secret sharing for mixed-mode and fixed-point secure computation," in *Advances in Cryptology – EUROCRYPT 2021*, 2021.

[5] E. Boyle, N. Gilboa, and Y. Ishai, "Function secret sharing," in *Annual international conference on the theory and applications of cryptographic techniques*. Springer, 2015.

[6] O. Catrina and S. De Hoogh, "Improved primitives for secure multiparty integer computation," in *Security and Cryptography for Networks*, 2010.

[7] D. Chaum, C. Crépeau, and I. Damgard, "Multiparty unconditionally secure protocols," in *Proceedings of the Twentieth Annual ACM Symposium on Theory of Computing*, 1988.

[8] J. Friginal, S. Gambs, J. Guiochet, and M.-O. Killijian, "Towards privacy-driven design of a dynamic carpooling system," *Pervasive and Mobile Computing*, 2014.

[9] D. Gale and L. S. Shapley, "College admissions and the stability of marriage," *The American mathematical monthly*, 1962.

[10] P. Hallgren, C. Orlandi, and A. Sabelfeld, "Privatepool: Privacy-preserving ridesharing," in *2017 IEEE 30th Computer Security Foundations Symposium (CSF)*, 2017.

[11] Y. He, J. Ni, X. Wang, B. Niu, F. Li, and X. Shen, "Privacy-preserving partner selection for ride-sharing services," *IEEE Transactions on Vehicular Technology*, 2018.

[12] A. Hegde, N. Koti, V. B. Kukkala, S. Patil, A. Patra, and P. Paul, "Attaining god beyond honest majority with friends and foes," in *International Conference on the Theory and Application of Cryptology and Information Security*, 2022.

[13] J. Huang, Y. Luo, S. Fu, M. Xu, and B. Hu, "pride: Privacy-preserving online ride hailing matching system with prediction," *IEEE Transactions on Vehicular Technology*, 2021.

[14] J. Huang, Y. Luo, M. Xu, B. Hu, and J. Long, "pshare: Privacy-preserving ride-sharing system with minimum-detouring route," *Applied Sciences*, 2022.

[15] Hurriyet Daily News, "Istanbul taxi drivers hunt down, beat up Uber drivers as tensions rise," https://www.hurriyetdailynews.com/istanbul-taxi-drivers-hunt-down-beat-up-uber-drivers-as-tensions-rise-128443, 2018.

[16] A. Ibarrondo, H. Chabanne, and M. Önen, "Funshade: Functional secret sharing for two-party secure thresholded distance evaluation," *Cryptology ePrint Archive*, 2022.

[17] B. Karmakar, N. Koti, A. Patra, S. Patranabis, P. Paul, and D. Ravi, "Asterisk: Super-fast mpc with a friend," in *2024 IEEE Symposium on Security and Privacy (SP)*, 2024.

[18] V. Kolesnikov, R. Kumaresan, M. Rosulek, and N. Trieu, "Efficient batched oblivious prf with applications to private set intersection," in *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, 2016.

[19] N. Koti, V. B. Kukkala, A. Patra, and B. Raj Gopal, "Pentagod: Stepping beyond traditional god with five parties," in *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, 2022.

[20] D. Kumaraswamy, S. Murthy, and S. Vivek, "Revisiting driver anonymity in oride," in *Selected Areas in Cryptography*, R. AlTawy and A. Hülsing, Eds. Cham: Springer International Publishing, 2022, pp. 25–46.

[21] Q. Li, H. Wu, and C. Dong, "A privacy-preserving ride matching scheme for ride sharing services in a hot spot area," *Electronics*, 2023.

[22] F. Martelli and M. E. Renda, "Enhancing privacy in ride-sharing applications through pois selection," in *2022 IEEE 23rd International Symposium on a World of Wireless, Mobile and Multimedia Networks (WoWMoM)*, 2022.

[23] McKinsey and Company, "https://www.mckinsey.com/features/mckinsey-center-for-future-mobility/mckinsey-on-urban-mobility/snapshot-of-the-european-car-sharing-market," 2022.

[24] D. Morales, I. Agudo, and J. Lopez, "Private set intersection: A systematic literature review," *Computer Science Review*, 2023.

[25] S. Murthy and S. Vivek, "Passive triangulation attack on oride," in *Cryptology and Network Security*, A. R. Beresford, A. Patra, and E. Bellini, Eds. Cham: Springer International Publishing, 2022, pp. 167–187.

[26] NortonLifeLock, "Uber Announces New Data Breach Affecting 57 million Riders and Drivers," http://lifelock.norton.com/learn/data-breaches/uber-data-breach-affects-57-million-rider-and-driver-accounts, 2020.

[27] E. Pagnin, G. Gunnarsson, P. Talebi, C. Orlandi, and A. Sabelfeld, "Toppool: Time-aware optimized privacy-preserving ridesharing," *Proceedings on Privacy Enhancing Technologies*, 2019.

[28] A. Patra, T. Schneider, A. Suresh, and H. Yalame, "{ABY2. 0}:

Improved {Mixed-Protocol} secure {Two-Party} computation," in *30th USENIX Security Symposium (USENIX Security 21)*, 2021.

[29] D. Sánchez, S. Martínez, and J. Domingo-Ferrer, "Co-utile p2p ridesharing via decentralization and reputation management," *Transportation Research Part C: Emerging Technologies*, 2016.

[30] TechCabal, "My driver the stalker," https://techcabal.com/2023/09/15/my-driver-the-stalker/, 2023.

[31] The Wall Street Journal, "https://www.wsj.com/articles/uber-to-pay-148-million-penalty-to-settle-2016-data-breach-1537983127," 2018.

[32] thejournal.ie, "West Dublin gang using hailing apps to target older taxi drivers," https://www.thejournal.ie/west-dublin-taxi-robbery-4420178-Jan2019/, 2019.

[33] N. Trieu, "Github - osu-crypto/BaRK-OPRF: Efficient Batched Oblivious PRF with Applications to Private Set Intersection (CCS 2016)," https://github.com/osu-crypto/BaRK-OPRF, 2016.

[34] F. Wang, H. Zhu, X. Liu, R. Lu, F. Li, H. Li, and S. Zhang, "Efficient and privacy-preserving dynamic spatial query scheme for ride-hailing services," *IEEE Transactions on Vehicular Technology*, 2018.

[35] X. Wang, A. J. Malozemoff, and J. Katz, "EMP-toolkit: Efficient MultiParty computation toolkit," https://github.com/emp-toolkit, 2016.

[36] Q. Xu, H. Zhu, Y. Zheng, J. Zhao, R. Lu, and H. Li, "An efficient and privacy-preserving route matching scheme for carpooling services," *IEEE Internet of Things Journal*, 2022.

[37] H. Yu, X. Jia, H. Zhang, X. Yu, and J. Shu, "Psride: Privacy-preserving shared ride matching for online ride hailing systems," *IEEE Transactions on Dependable and Secure Computing*, 2021.

[38] H. Yu, H. Zhang, X. Yu, X. Du, and M. Guizani, "Pgride: Privacy-preserving group ridesharing matching in online ride hailing services," *IEEE Internet of Things Journal*, 2021.

[39] J. Zhang, L. Wang, X. Hu, R. Li, and S. Zheng, "Privacy-preserving online ride-hailing service system based on taking the intersection of private sets of points of interest," in *2023 International Conference on Mobile Internet, Cloud Computing and Information Security (MICCIS)*, 2023.

[40] Q. Zhou, Z. Zeng, K. Wang, and M. Chen, "Privacy protection scheme for the internet of vehicles based on private set intersection," *Cryptography*, 2022.