Paper **Six**

PrivatePool

Privacy-Preserving Ridesharing

Per Hallgren, Claudio Orlandi and Andrei Sabelfeld

Location-based services have seen tremendous developments over the recent years. These services have revolutionized transportation business, as witnessed by the success of Uber, Lyft, BlaBlaCar, and the like. Yet from the privacy point of view, the state of the art leaves much to be desired. The location of the user is typically shared with the service, opening up for privacy abuse, as in some recently publicized cases. This paper proposes PrivatePool, a model for privacy-preserving ridesharing. We develop secure multi-party computation techniques for endpoint and trajectory matching that allow dispensing with trust to third parties. At the same time, the users learn of a ride segment they can share and nothing else about other users' location. We establish formal privacy guarantees and investigate how different riding patterns affect the privacy, utility, and performance trade-offs between approaches based on the proximity of endpoints vs. proximity of trajectories.

Published in the IEEE 30th Computer Security Foundations Symposium
CSF 2017, Santa Barbara, CA, USA, August 21-25, 2017
Revised version

1 Introduction

Location-based services (LBS) have seen tremendous developments over the recent years. These services have revolutionized transportation business, as witnessed by the success of Uber [53], Lyft [31], and the like. These technologies leverage the idea of *ridesharing*. The up-and-coming service BlaBlaCar [2] epitomizes the simplicity of ridesharing: a user may advertise that they are traveling between two points, but can take a passenger user or ride with another user. The obvious benefit for the users is to reduce the cost of travel.

Motivation Yet from the privacy point of view, the state of the art leaves much to be desired. The location of the user is typically shared with the service, opening up for privacy abuse. The ridesharing app Uber, connecting passengers with private drivers, has been the subject of much privacy debate. Uber and its employees have been allegedly involved in privacy-violating activities from stalking journalists and VIPs to tracking one-night stands [1].

Beyond ridesharing, digitalization of transportation systems and development of self-driving cars [54] opens up further opportunities, which is expected to drive collection of unprecedented amounts of data. With the advent of self-driving cars, such as the Tesla Model 3 [55], the market will sport a fully digitalized car with autopilot functionality in the hands of consumers. In the near future consumers will hold technology that can provide automated commuting services. To optimize such services, the daily itinerary of end users is needed, such that users who travel to and from the same approximate area can be transported together. When this occurs, users are expected to provide the service provider with their private location data. This raises alarming privacy concerns.

Overall, the sensitivity of the private location information in ridesharing applications poses a major challenge: *how to preserve privacy without hampering the functionality of the ridesharing services?*

Privacy-preserving ridesharing We propose PrivatePool, a novel model for privacy-preserving ridesharing. Our goal is to provide the unhampered functionality of normal ridesharing applications without compromising privacy through means of *secure multi-party computation (SMC)* [3, 11], where participants can jointly compute a function based on private inputs.

This work focuses on *flexible* ridesharing, similar in that respect to BlaBlaCar we accommodate users that are willing to divert from their original path. The economical and environmental savings have impact on whether ridesharing is feasible.

This can be manifested by a maximum deviation from each user's path and a minimum overlap of the trajectories for desirable ridesharing. Intuitively it is harder for the user to deviate in the middle of the trip than at the beginning or end, which means that the deviation allowed is not constant. For instance, if a ride is shared over a larger distance, such as between cities, it would likely be acceptable for the users to use public transport to coordinate within the origin or destination city, but a rendezvous near the center of the trip is much more restricted.

In contrast to BlaBlaCar, however, we are interested in ridesharing with no trust to third parties. We envision that our approach will accommodate a decentralized version for ridesharing services that will break away from the traditional full disclosure of user location. As such, our work is a step in the direction of building theoretical foundations for such a decentralized service, that – in the long run – has potential to lead to practical systems for solving problems like the ones publicized in [1].

To the best of our knowledge, this paper presents the first model for privacy-preserving ridesharing. To accommodate flexible privacy-preserving ridesharing, we draw on two key building blocks: one is based on *proximity* of the journeys' start- and endpoints and the other one is based on *intersection* of the routes between the start- and endpoints.

There has been much recent work on secure multi-party computation of *proximity testing* [57, 48, 47, 7, 32, 35, 45, 15, 42], where the goal is to compute whether two parties are close to each other without revealing their relative distances and positions to each other or to any third party. However, proximity testing by itself does not solve the ridesharing problem. First, we need a method to securely extend proximity testing to yield a match when *both* the respective start and endpoints are within desirable proximity. A naïve application of proximity protocols on the start/endpoints would violate privacy: if for two rides the start points are far but the endpoints are close, the naïve approach would reveal the proximity of the endpoints in the absence of a shared ride. Second, flexible ridesharing in some scenarios necessitates considering trajectories and not only the start- and endpoints.

Indeed, since we are interested in flexible ridesharing, the start/endpoints might be actually far enough while there is still a large segment of the route that can be shared by the users. A typical example is an intercity ride that starts and ends in different parts of the origin and destination cities. In this scenario, we draw on the *private set intersection (PSI)* [6]. By applying PSI to sets representing trajectories/routes, we can compute whether there is a sufficient overlap to warrant a shared ride. Again, PSI by itself does not solve the ridesharing problem either. Recall that we

only want to reveal a match when there is a sufficient segment that overlaps between the users' trajectories. This motivates the need for a *threshold private set intersection (T-PSI)* protocol, enabling us to achieve private and flexible trajectory matching. As a side remark, we note that the term threshold set intersection, or over-threshold set-union, has also been used in the literature to describe the scenario when one wants to disclose all elements among n users' private input sets which occurs in the input of at least a threshold number t users [24, 25]. This scenario is different from the one considered in the paper.

Finally, achieving the above goals would only make sense if it yields techniques that provide utility, respect privacy, and have feasible computation overhead. We thus set out to evaluate these techniques on a collection of realistic ridesharing patterns. In the evaluation, it is also our goal to compare our approach with respect to generic secure multi-party techniques, with the focus on the state of the art technique of garbled circuits [56].

Contributions We develop the first model for privacy-preserving ridesharing. On the policy side, the model accommodates flexible ride sharing, allowing to be parametric in how long the users are willing to travel to meet up for a shared ride and for how long a ride they require in a successful match. On the enforcement side, we develop two independently interesting enforcement mechanisms. It enables ride matching by both the proximity of start- and endpoints and by trajectory matching. For endpoint matching we build on top of existing work utilizing additively homomorphic encryption to create a privacy-preserving protocol. At the core of the trajectory matching, we design a novel threshold private set intersection (T-PSI) protocol, for which we establish rigorous privacy guarantees. The main technical contribution is the definition and construction of a so called *threshold key encapsulation mechanism* (T-KEM). We present an overview of realistic ride patterns and evaluate our mechanisms with respect to these patterns. Benchmarking against the patterns demonstrates that we benefit from the generality of our approach: start/endpoint matching and trajectory matching excel on different patterns. At the same time, our benchmarks also show that our techniques are preferable over a generic approach, as implemented by garbled circuits.

Limitations While the cryptographic techniques evaluated can be called practical out-of-context, a fully-fledged ridesharing system needs more work before it is useful in practice. On the one hand, practically-oriented research results are needed to show how to use SMC in general without information leakage from a running system. As SMC works on the application layer, it is oblivious to information on

other levels such as IP addresses etc. Lacking an outlook for a secure full system, in this first effort for privacy in ridesharing applications several details which could be privacy-sensitive are left for future work. This includes the time of the ride and the identity of the users.

On the other hand, further foundational work is needed in terms of scalability to large numbers of users. The state of the art is making great leaps in this direction, such that is now possible to efficiently compute a fixed function of many users [21]. However, state-of-the art SMC protocols are only efficient for a limited number of users for cases like ours where each party wants to evaluate a different function, as the question "who can I share a ride with?" is context-sensitive. This limitation is shared with the entire line of work on privacy-preserving location proximity protocols (e.g., [57, 35, 45, 15]).

Overview The rest of the paper is organized as follows. Section 2 presents ridesharing concepts, such as feasibility, and discusses ridesharing patterns. Section 3 explains the concepts of proximity testing and private set intersection, key building blocks in our approach. Section 4 present a novel mechanism for threshold private set intersection, which allows us to parameterize the privacy of trajectory matching. Section 5 details the threshold key encapsulation mechanism that lies at the heart of our cryptographic construction. Section 6 presents the experiments that illustrate that different patterns benefit from the different mechanisms in our approach. Section 7 discusses related work. Section 8 concludes.

2 Ridesharing concepts

This section presents the basics of our ridesharing model and characterizes ridesharing feasibility. Based on these, we discuss different patterns where ridesharing is feasible, of which two are studied further in Sections 3, 4 and 6.

2.1 Modeling ridesharing

The model considers users who set out on *trips* as defined in Definition 1, traveling from one point to another. To describe such trips, we consider the map as an undirected, unweighted graph $G = (V, E)$, where V is the set of vertices and $E \subseteq V \times V$ is the set of edges. An edge e can be imagined as being a two-way road section, connecting two coordinates.

The model does not include the identities of the users or the time frame in which a user is willing to participate in ridesharing, and does not say anything about

whether or not a service provider is involved in the exchange. Only the abstract view of the two parties exchanging location data is considered, leaving protection of other private information and detailing what data is shared with service providers and third parties for future work.

Definition 1 (Trip). *Given a graph $G = (V, E)$, a* trip T *is an acyclic sequence of consecutive vertices $v_i \in V$, where $v_s = v_0$ is the origin and $v_f = v_{|T|-1}$ is the destination, such that $(v_i, v_{i+1}) \in E$ for all $i \in \{0, \ldots, |T| - 2\}$.*

A set of consecutive vertices in a trip T is called a *segment* as per Definition 2.

Definition 2 (Segment). *Given a graph $G = (V, E)$, a* segment S *of a trip T in G is an acyclic sequence of consecutive vertices $v \in V$, such that S is a subsequence of T.*

Users conduct trips by traversing the graph using the shortest path from their origin to their destination. The set of vertices visited when traversing the shortest path is called the user's *trajectory*. Users are considered as traveling with constant speed for the scope of this work, such that both the spatial and temporal cost of traversing a road section is equivalent. How the graph and the trajectories are constructed are out of the scope of this work. If the application need to take e.g. traffic congestion into account, the length metric should be updated to accommodate this.

When utilizing ridesharing, users may deviate from their trajectory, for the purpose of one user aligning their trip to the other user's trajectory. That is, the first user A pays an extra cost, extending their trip such that it includes a part of another user B's trajectory, while B travels exactly along their trajectory. Given any protocol implementing ridesharing, one can switch roles and rerun the protocol to symmetry between A and B. Users are restricted to sharing a ride during a single segment, which leads to five segments of significance during A's trip, called the *ridesharing segments* of the trip, as illustrated in Figure 1. These segments are:

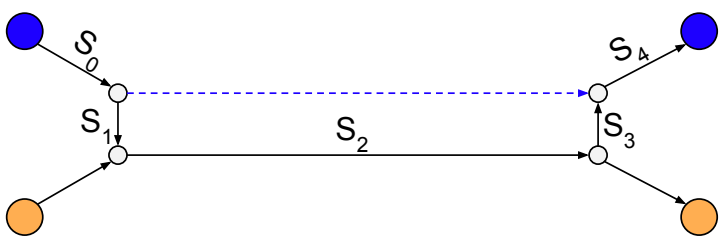

Fig. 1. The five common ridesharing segments

1. S_0, the start segment, is a part of the trajectory and includes at least $v_s \in T$.
2. S_1, where the user is traveling to the ridesharing segment, is a part of neither user's trajectory.
3. S_2 is the ridesharing segment, the maximal segment where the two users can share a ride.
4. S_3 is the counterpart of S_1, where the user is traveling from the ridesharing segment.
5. S_4, similarly to S_0, is a part of the trajectory, where the last vertex is $v_f \in T$.

In short, S_0 and S_4 are parts of the trajectory, which would be also without ridesharing. S_1 and S_3 are the additional path that the user travels, leaving their trajectory and aligning with the other user's trajectory. Finally, S_2 is the part of the trip where the users share a ride. To make the representation of a trip more concise, Definition 3 specifies the length of a segment S as $l(S)$, assuming that we have a Euclidean distance defined for each edge. Note that $l(S) \neq |S|$.

Definition 3 (Segment length). *Given a segment S for some trip through a graph $G = (V, E)$, let $l(S) = \sum_{i=0}^{|S|-2} d\left(S[i], S[i+1]\right)$, where $S[j]$ is the jth vertex in S and $d(p_1, p_2)$ is the Euclidean distance between the two points p_1 and p_2.*

2.2 Ridesharing feasibility

When ridesharing comes with a low enough cost and high enough benefit for both parties, it is called *feasible* ridesharing. Feasibility is modeled with two parameters.

First, an upper limit as to how much a user is willing to extend the trip (i.e., deviate from their trajectory) before and after S_2. As previously highlighted, this distance may depend on how far the user has traveled from their endpoints (i.e., how long S_0 and S_4 are). This is captured using a deviation function $\Delta_T(v)$, where T is the user's trajectory and $v \in T$ is a vertex along the trajectory. The evaluation of $\Delta_T(v)$ at a vertex v gives us the user's *deviation limit* at that stage of their trip, which is a measure of how flexible the user is. Ridesharing is feasible only when $l(S_1) < \Delta_T(v_0) \wedge l(S_3) < \Delta_T(v_1)$, where S_1 and S_3 are ridesharing segments of T, $v_0 \in T$ is the first vertex in S_1 and $v_1 \in T$ is the last vertex in S_3.

Secondly, we model a lower limit of the length of the ridesharing segment as a distance threshold, called t, simply called the *threshold*. This enables the model to be used in cases where the trip's economic or environmental cost is to be reduced by some factor for ridesharing to be feasible. Concisely, ridesharing is feasible only if $l(S_2) > t$.

Finally, the formalization of a feasible ridesharing scenario is given in Definition 4. Using $\Delta(\cdot)$ and t, most preferences of a user can be modeled. Note that no restriction on $\Delta(\cdot)$ has been made, it could be an arbitrary function, and may be defined on a per-user and per-query basis, such that A specifies the deviation function while making each query.

Definition 4 (Ridesharing feasibility). *For any fixed threshold t and deviation function Δ, given two trajectories P_A and P_B for users A and B in $G = (V, E)$, ridesharing is feasible for A along a segment $S = \{p_s, \ldots, p_f\}$ of P_B if and only if $l(S) > t$ and there exist two points $P_A[i]$ and $P_A[j]$, with $i < j$, such that:*

$$d(P_A[i], p_s) < \Delta_{P_A}(P_A[i])$$
$$\wedge\, d(P_A[j], p_f) < \Delta_{P_A}(P_A[j])$$

Another important measure is how much A's total trip is extended by utilizing ridesharing. This is bounded by Δ, as per Theorem 1.

Theorem 1 (Maximum trip extension). *If ridesharing is feasible for A for a trip T with ridesharing segments $S_i, i \in \{0..4\}$ and where $v_0 \in T$ is the first vertex in S_1 and $v_1 \in T$ is the last vertex in S_3, the total trip will have a length shorter than $l(T) + 2(\Delta_T(v_0) + \Delta_T(v_1))$.*

Proof. Let the skipped segment of A's original trip be $S_s = T \setminus (S_0 \cup S_4)$. Now, assume by contradiction that the total trip is extended by at least $2d$ with $d = \Delta_T(v_0) + \Delta_T(v_1)$ such that $2d \leq l(S_1) + l(S_2) + l(S_3) - l(S_s)$, which gives us Equation 1.

$$l(S_s) + 2d \leq l(S_1) + l(S_2) + l(S_3) \tag{1}$$

By construction, from the first vertex of S_2 to the first vertex of S_s, the maximum distance is $\Delta_T(v_0)$. Similarly, the distance from the last vertex of S_s to the last vertex of S_2 is at most $\Delta_T(v_1)$, which gives Equation 2.

$$l(S_1) + l(S_3) < d \tag{2}$$

Using Equation 2, we we can substitute $l(S_1)$ and $l(S_3)$ in Equation 1, and see that $l(S_s) + 2d < l(S_2) + d$. Now we can apply Equation 2 again, on the left-hand side, to arrive at $l(S_s) + l(S_1) + l(S_3) + d < l(S_2) + d$, or simply $l(S_1) + l(S_s) + l(S_3) < l(S_2)$. This implies that the shortest path from the first vertex of S_2 to the last vertex of S_2 is via S_1, S_s, and S_3. Therefore, S_2 is not a part of any shortest path in G, and thus not a part of B's trajectory. □

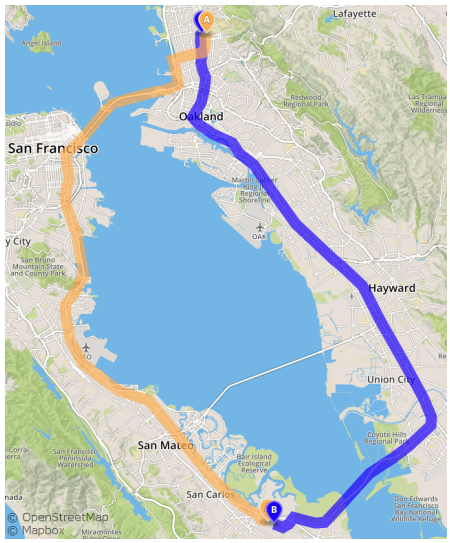

Fig. 2. The paths barely intersect while the endpoints are close

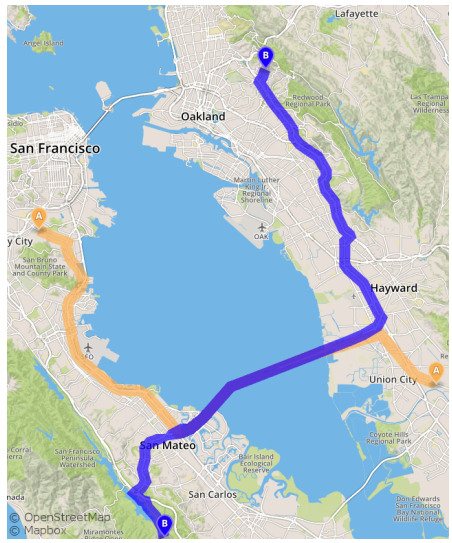

Fig. 3. The paths intersect substantially while the endpoints are far

2.3 Ridesharing patterns

This section discusses distinct *ridesharing patterns* which can be captured with the model.

The *proximity-based* pattern is a convenient ridesharing pattern. In this pattern, there is only a short commute before and after the ridesharing segment, making the effort for changing means of transport low. Such scenarios are easy to find in sparsely connected areas, which is a common situation close to bodies of water, such as rivers with few bridges or lakes. One example is the San Francisco Bay Area as depicted in Figure 2 which shows two trips from Berkley to Redwood City. In this example, the trajectories intersect only an insignificant part of the trip, but where ridesharing clearly is possible as the origins and destinations are only 10 minutes walking distance from each other.

On the other hand, many applications are likely interested in detecting ridesharing patterns where the traveled distance is minimized, which is captured by the *intersection-based* pattern. Consider for instance the one depicted in Figure 3, where the extra distance traveled to enable ridesharing is not relevant. Both users would either way travel along the ridesharing segment. In this case the endpoints are far apart, but the overlap between the two trajectories is roughly half the trip, which is

a reasonable distance to utilize ridesharing. This pattern is likely to be found if users travel via common junction points, e.g. along major highways.

Both the western and the eastern route are acceptable to each party in Figure 2. This highlights a case that is not implicitly captured by the described model. In the case when B could go both the western and the eastern route, but where the eastern one is negligibly shorter, but A cannot take the eastern route (A is e.g. going from downtown San Francisco to Redwood City). However to capture such cases it suffices to do a separate matching for each acceptable route. An alternative approach is to let A and B meet in the middle, as if both are willing to deviate by some distance, more rides can be shared. This setting can be trivially captured by adding B's deviation limit to that of A for every vertex.

While the above patterns are common, we note that there are yet other scenarios, for instance several scenarios would match a *hybrid* pattern of the above two as depicted in Figure 4. In this case the origins (similar for the destinations) are close but an overlap occurs much later. Correspondingly for the Bay Area example in Figure 2, assuming the origins are the south endpoints, if either user would have continued their journey farther north, only the origin endpoints would be closer than the deviation limit, and the intersection would be smaller than t (for reasonable parameters).

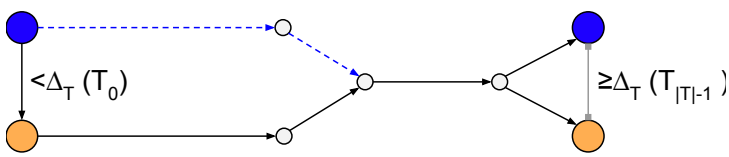

Fig. 4. Ridesharing pattern with small overlap and far endpoints

Having presented the ridesharing concepts and patterns, we are now ready to describe our mechanisms for achieving privacy-preserving ridesharing.

3 Privacy-preserving ridesharing

This section details how to privately detect the concrete patterns given above. As will be shown, the two patterns depicted in Figure 5 and Figure 6 are amenable to SMC. The first pattern can be realized using privacy-preserving proximity testing with a few modifications and the second pattern is exactly the use case for private set intersection.

3.1 Privately detectable patterns

Given the patterns above, the question remains how to design an algorithm to efficiently detect ridesharing scenarios – and further, to design algorithms that are amenable to SMC. The goal of privacy-preserving ridesharing is to disclose the ridesharing segment only if ridesharing is feasible, and no information otherwise.

Two of the distinct patterns turn out to be susceptible to SMC techniques. These two patterns result in enforcing either of the two criteria $t \geq l(T) - \Delta_T(T_0) - \Delta_T(T_{|T|-1})$ and $\Delta_T(\cdot) = 0$ for proximity-based and intersection-based patterns, respectively (where again t is the threshold and Δ_T is the deviation function). That is, in the first case the shared trip is as long as the full trip, except for possibly the cost of A traveling to and from B's starting point as illustrated in Figure 5. In the second case, the shortest paths intersect, and there's no extra distance to travel to enable ridesharing as illustrated in Figure 6.

A straightforward "bruteforce" algorithm that attempts finding the longest possible ridesharing segment is to find the first vertex $v_f \in A$ which is closer than $\Delta_{P_A}(v_f)$ to any point on B's trajectory and last vertex $v_l \in A$ which is closer than $\Delta_{P_A}(v_l)$ to any point on B's trajectory. Ridesharing is then feasible if between v_f and v_l if $d(v_f, v_l) > t$. However, such an algorithm does not have any apparent efficient implementation using SMC.

Instead, we now outline algorithms for proximity-based and intersection-based ridesharing patterns, which are amenable to SMC. Proximity-based patterns are detected by checking that the start and endpoints are close. As seen in Section 3.2, this can be achieved with SMC with private proximity testing. Intersection-based patterns can be detected by computing the intersection $I = P_A \cap P_B$, and conclude that ridesharing is feasible when $l(I) > t$. As seen in Section 4, this is achieved through our new primitive T-KEM.

For the hybrid pattern (and others), there appear to be no apparent efficient privacy-preserving solution. One could imagine detecting proximity endpoints and subsequently checking for intersection. However, such an approach would leak endpoint proximity even if it is impossible to share a ride. Of course, generic SMC solutions could be used to achieve the sought functionality, but as we will see in Section 6.2 such solutions are as of yet not efficient enough for most applications.

How many scenarios do not match the patterns detailed above, and how relevant they are for practical applications, is hard to judge without empirical data. To shed some light on the matter, we have carried out experiments on real-world data from the New York City taxi services, as detailed in Section 6.1. The experiments show that using both approaches can achieve up to 92% of the effectiveness of the entire model,

and that both endpoint-based matching and intersection-based matching excel on different rides.

3.2 Proximity-based Ridesharing

For the first pattern, depicted in Figure 5, the users A and B want to check whether their start- and endpoints are closer than a fixed limit r ($\Delta_A(\cdot) = \Delta_B(\cdot) = r$). To check if two points are close is often called *proximity testing*, for which there are several privacy-preserving solutions with different strengths and weaknesses, such as the amount of privacy provided, the number of roundtrips, and performance requirements [35, 45, 15].

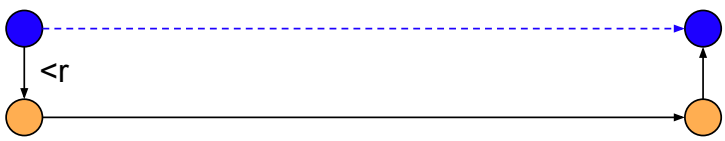

Fig. 5. Ridesharing pattern with maximum ridesharing segment

However, in addition to checking the proximity of two endpoints, a construction is needed that allows to disclose a result only if both endpoints are close to each other. This is not trivial to achieve with an arbitrary proximity testing solution. Luckily, there are several solutions based on homomorphic encryption, where it's common to represent a positive response as encryption of 0, and a negative response as an encryption of a uniformly random value. This enables the two results to be multiplied to create an "or" operator, or added to create an "and" operator. Thus, when the start and endpoint results have been computed, they can be added to produce the final result.

We pick the InnerCircle approach by Hallgren et al. [15] as a starting point because it allows for the lowest round-trip time amongst protocols in the literature. The protocol is also among the better when it comes to other properties such as number of roundtrips and the level of privacy provided. The construction requires an unfortunate restriction in terms of precision, where discretization up to 10 or 100 meters is required. This imprecision is likely not mission-critical in a ride-sharing application and thus the boost in performance as compared to other approaches outweighs this drawback.

As with other recent efficient approaches [35, 45], it works with *additively* homomorphic encryption utilizing well-studied cryptosystems. Further motivating this choice is that the line of work by Hallgren et al. is amenable to composition while protecting against malicious adversaries [14] with few modifications.

The construction is briefly detailed here to highlight the necessary adjustments. The user A holds the private key, for which B has the corresponding public key. A queries B for proximity by issuing a request with a triplet $(x_A, y_A, x_A^2 + y_A^2)$. B computes the squared euclidean distance as:

$$D = x_A^2 + y_A^2 + x_B^2 + y_B^2 - 2\,(x_A x_B + y_A y_B)$$

B can do all of these computations in the encrypted domain as they are only linear operations and thus supported by an additively homomorphic encryption system. Next, B needs just to compare D with the radius value r, which is done by encoding the comparison in a set

$$\{(D - i)\,\rho_i | i \in \{0..r^2\}\}$$

Where each ρ_i is an independent random number in the plaintext space. The set is sent to A in a random order, such that A only can deduce whether $\exists i < r^2 : i = D$, which is equivalent to $D < r^2$.

Now back to the ridesharing setting, where a procedure is executed in three steps. First, B computes an array which encodes the proximity result. Secondly, for B to be able to combine the proximity results for the origin and destination, the arrays need to reduced to a single results. This can be done using successive "or" operations, via multiplications as mentioned earlier. Using additively homomorphic means that multiplications of ciphertexts cannot be computed locally. A common workaround is to send blinded values to A, who can decrypt, multiply, and encrypt the result before sending it back [27]. The blinding is then removed and B now has two ciphertexts encrypting either 0 or a uniformly random number. Though the multiplications incur an overhead, consecutive outsourced multiplications will introduce only a logarithmic number of round trips [16]. As a third and final step, B executes the "and" operation of the two ciphertexts, and the final result is sent back to A.

3.3 Intersection-based Ridesharing

The second pattern, as shown in Figure 6, is for users who want to minimize the deviation and find a trip such that $\Delta_T(\cdot) = 0$. That is, the user is looking for trips

that overlap perfectly with their own trip, which as highlighted previously is solvable using PSI. Recent research has yielded ad-hoc solutions for PSI [39] which are highly efficient. However, applying PSI directly would always disclose the intersection, regardless of its size. For instance, with two orthogonal trajectories which only intersect at a single point, disclosing this point does not provide information useful for ridesharing, but leaks privacy-sensitive location-information. Thus, as outlined in Section 4, we define a novel ad-hoc protocol for threshold-conditioned PSI which does not have these fallacies.

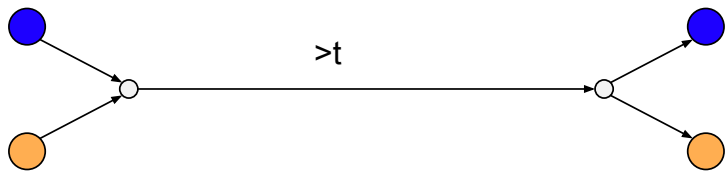

Fig. 6. Ridesharing pattern with minimum cost

There are also efficient solutions utilizing generic circuits to achieve PSI [17]. These are by construction amenable to composition with other methods, and are therefore an interesting comparison to our novel scheme. We outline a performance comparison in Section 6.2.

3.4 Practical applicability of SMC

The above mentioned techniques share restrictions that apply to SMC in general. As highlighted in Section 1, these are both in terms of scalability in the number of users, and in terms of complete system-wide enforcement.

There is much work to be done before privacy of location-data can be guaranteed when considering a running application on existing operating systems for mobile devices. This can be seen through various side-channel attacks on location data [29, 49, 34, 38, 33], including vectors such as power consumption and carrier signal strength. While anonymization techniques such as onion routing may be used to escape threats from internet infrastructure providers, cellular network providers provide further complications which are harder to tackle with existing hardware. Theoretical models for security, such as the one used in this work, try to minimize the trust that is needed to be placed in a party. Instead of the user completely trusting the service provider, one allows the user to place trust in technology/cryptography.

Though there are no current solution that frees the user from trusting anything but cryptography as per the above issues, reducing the amount of trust the user has to place on a service provider or other users is still of tangible value.

In terms of scalability, there are no solutions that are asymptotically comparable to that of using a trusted third party. While there are many solutions for secure *multiparty computations*, such that any number of parties can jointly compute a function without the communication, the restriction is to compute a *single* function. These solution are very useful if the function has the same output no matter who issues the query, such as services for auctions or elections, or when only one party is interested in the output, perhaps according to a business agreement. However, when the question is no longer "Who made the highest bid?" or "Who had the most votes?", but rather "Can I share a ride with someone?", the query is not so easily answered. The solutions proposed in this work are both peer-to-peer, which means that a user needs to communicate with every other party to check for ridesharing opportunities, and thus requires $\mathcal{O}(n)$ parties to communicate. This may make the solution hard to apply in various scenarios, but even with recent and very promising works focusing specifically on scalability, each party needs to communicate for every evaluation of a function [21, 13], which again requires $\mathcal{O}(n)$ communication.

There are many more issues to consider when trying to build a system utilizing SMC. One such issue is secure key distribution. We leave this issue, as well as identity management in general, out of scope and subject to future work. Further, the adversary considered in this work is passive, while many applications need to be secure against active adversary. Both the technique for endpoint matching and for set intersection can be adjusted to give heightened security against active adversaries at the cost of performance [14, 43]. As seen in Section 6.2, the choice of a weaker adversarial model allows us to cater for many applications, while this may not be the case if using a slower (though more secure), version.

4 Threshold PSI

In this section we describe our solution to the problem of securely evaluating the threshold private set intersection (T-PSI) between the sets of two parties, called the *sender* and the *receiver*, in the presence of a passive adversary and static corruption. The sender and receiver have as inputs two sets A and B respectively, both of size n. At the end of the protocol the receiver should learn $(A \cap B)$ if $|A \cap B| \geq t$ for some predetermined threshold t or nothing otherwise.

The sender learns nothing. Though the aim with this work is privacy-preserving ridesharing, we hypothesize that the T-PSI construction is useful in many other areas. For instance, it could be applied to dating applications; If Alice requires her future husband to share at least two of her hobbies, it makes little sense to reveal any of Alice's hobbies to Bob if they share only one.

Before proceeding further, let us outline some basic concepts as often used in the SMC literature in Definition 5, Definition 6 and Definition 7.

Definition 5 (Negligible functions). *A function* $\epsilon : \mathbb{N} \to \mathbb{R}$ *is* negligible *if*

$$\forall\, c \in \mathbb{N}.\ \exists\, n_c \in \mathbb{N}.\ \forall_{n \geq n_c}\ |\epsilon(n)| \leq n^{-c}$$

That is, ϵ *decreases faster than the inverse of any polynomial.*

Definition 6 (Indistinguishability). *The two random variables* $X(n, a)$, $Y(n, a)$ *(where* n *is a security parameter and* a *represents the inputs to the protocol) are called* computationally indistinguishable *and denoted* $X \overset{c}{\equiv} Y$ *if for any probabilistic polynomial time (PPT) adversary* $\mathcal{A}$ *the function* $\delta(n)$ *is negligible:*

$$\delta(n) = |\Pr[\mathcal{A}(X(n, a)) = 1] - \Pr[\mathcal{A}(Y(n, a)) = 1]|$$

Definition 7 (Semantic Security). *A public key encryption scheme* E *is* semantically secure *or IND-CPA secure if the advantage of any PPT adversary of winning the below game against* $\mathcal{C}$ *is negligible. The game is won if the attacker outputs* $b' = b$.

1. $\mathcal{A}$ *outputs two different plaintexts* (m_0, m_1).
2. $\mathcal{C}$ *flips a random* $b \in \{0, 1\}$
3. $\mathcal{C}$ *outputs* $E(m_b)$
4. $\mathcal{A}$ *outputs b'*

Now for a brief outline of simulation-based proofs, for which our definition follows from common definitions of secure multi-party computation in the passive (or honest-but-curious) adversarial model [10, 30], but is here simplified for the case with two parties. For two parties Alice and Bob, where $Alice$ has inputs $\overrightarrow{x}$ and Bob inputs $\overrightarrow{y}$, the model formalizes the output of a protocol as $f(\overrightarrow{x}, \overrightarrow{y}) = (g(\overrightarrow{x}, \overrightarrow{y}), h(\overrightarrow{x}, \overrightarrow{y}))$. The function f is called the *functionality* of the protocol. The functions g and h are functions describing all outputs presented to Alice and Bob from the execution of the protocol, respectively.

Definition 8 (Privacy). Privacy *for deterministic functionalities hold when the overall knowledge of each party after the execution of the protocol, called the party's* view,

can be computed from the inputs and outputs of that party. This is called that the view can be simulated. *That is, for the two-party case with Alice and Bob as described above, one must show that:*

$$\{\mathcal{S}_{Alice}(\overrightarrow{x}, g(\overrightarrow{x}, \overrightarrow{y}))\} \stackrel{c}{\equiv} \{view_{Alice}(\overrightarrow{x}, \overrightarrow{y})\}$$
$$\{\mathcal{S}_{Bob}(\overrightarrow{y}, h(\overrightarrow{x}, \overrightarrow{y}))\} \stackrel{c}{\equiv} \{view_{Bob}(\overrightarrow{x}, \overrightarrow{y})\}$$

where $\mathcal{S}_{Alice}$ and $\mathcal{S}_{Bob}$ are the simulators for Alice and Bob, respectively.

4.1 Phasing&Co.

The last few years have seen a large improvement in the efficiency of protocols for PSI, most notably the Phasing protocol [40] and its improvements [39, 26][1]. For the sake of exposition we choose to abstract as much as possible the properties of the Phasing protocol to clearly communicate the main changes which are necessary to turn this into a T-PSI protocol. From a very high-level point of view, the Phasing protocol proceeds in two stages:

Phasing – Stage 1: Masks Generation In the first stage the sender and the receiver compute sets of random strings called *"masks"* based on their input sets. In particular, at the end of the first stage, the sender learns (for each element $a \in A$) a set of random masks $M_a = (m_1, \ldots, m_j)$. Similarly, for each $b \in B$ the receiver learns a set of random masks $M_b = (m_1, \ldots, m_k)$ under the constraint that, if $a = b$, then $|M_a \cap M_b| = 1$. At the same time, choosing masks of appropriate lengths ensures that if $a \in A$ but $a \notin B$ then $\forall b \in B$, $M_a \cap M_b = \emptyset$. The first stage of Phasing can be used directly in our final construction, to reuse the subroutine we write for short:

$$\mathsf{Phasing}(A, B) \rightarrow (U, (V, R_V))$$

Where $U = \{M_a | a \in A\}$ is output to the sender, and $V = \{M_b | b \in B\}$ and R_V is output to the receiver. R_V is a reverse-mapping needed by the receiver to find which element in B is masked by a given mask. We define a function to find the reverse mapping, $\mathsf{RevMask}(\mathsf{M_b}, \mathsf{R_V}) \rightarrow b$, such that for any element $b \in B$ it returns the item b masked by M_b.

[1] The term "Phasing" first appeared in [39] but here we use it as a general term for the whole family of protocols.

Phasing – Stage 2: Computing the Intersection In the second stage of the Phasing protocol the sender sends the set $U = \{M_a | a \in A\}$ to the receiver, who can then compute the intersection of U and each M_b to determine whether b is in the intersection. We define the procedure Isect to reuse in the final protocol:

Procedure $\mathsf{Isect}(U, V, R_V)$
  $Z \leftarrow [\,]$
  for $M_i \in V$ **do** :
    if $U \cap M_i = 1$ **then**:
      $Z \leftarrow Z :: \mathsf{RevMask}(\mathsf{M_i}, \mathsf{R_V})$
  return Z

Security of Phasing For completeness, we show how to construct simulators for our high-level view of Phasing, called $\mathcal{S}_S^{\text{PHG}}$ and $\mathcal{S}_R^{\text{PHG}}$ for the both the sender and receiver, respectively. The claims follow from the proofs of the concrete instances of phasing protocols [40, 39, 26]. The privacy of the overall protocol follows from the following two properties about the first stage of Phasing:

- in the view of the receiver for the first stage the masks which are not in the intersection are computationally indistinguishable from uniform random strings.
- the sender does not learn anything about the set B of the receiver during the first stage. Formally, the view of the sender (and in particular $M_a \forall a \in A$) can be simulated without access to the input B.

The proof of the Phasing protocol [40] implicitly contains the proof of these two properties.

Since the sender does not receive any messages during the second stage and the first stage satisfies the property that it is possible to simulate M_a without access to B, then the protocol is secure against passively corrupt senders. Formally, there exists a simulator for the sender $\mathcal{S}_S^{\text{PHG}}(A) \rightarrow U$ that takes as input the input set A and produces a set of masks U which is indistinguishable from the view of the sender in a real protocol execution.

To argue the security of Phasing against a passively corrupt receiver, the authors construct a simulator that produces a set U to contain the desired intersection with V (consistent with the input B and the output V), and otherwise populates the set U with uniformly random strings. In a nutshell, the simulator for the receiver $(U, V) \leftarrow \mathcal{S}_R^{\text{PHG}}(B, Z)$ works as follows:

1. The simulator $\mathcal{S}_R^{\text{PHG}}(B, Z)$ parses its input and checks that $Z \subseteq B$ and aborts otherwise;

2. The simulator samples uniformly random masks from $\{0,1\}^\lambda$ for the receiver $V = \{M_b | b \in B\}$.
3. The simulator constructs the set of masks for the sender U in the following way: for every $b \in Z \cap B$, the simulator picks a random mask from M_b and adds it to U. Then, the simulator fills U with $n - |Z|$ random masks.

This simulation is indistinguishable from the view of the receiver in a real execution of the protocol. We now extract a property which is needed later in our construction and which is satisfied using $\mathcal{S}_R^{\text{PHG}}$ above:

Definition 9 (Phasing – Sender's Privacy). *We define a game between an adversary $\mathcal{A}$ and a challenger $\mathcal{C}$:*

1. *$\mathcal{A}$ outputs two sets (A, B)*
2. *$\mathcal{C}$ flips a random coin $b \in \{0,1\}$*
3. *If $b = 0$, $\mathcal{C}$ output $\mathsf{Phasing}(A, B)$*
4. *If $b = 1$, $\mathcal{C}$ outputs $\mathcal{S}_R^{P_{HG}}(B, A \cap B)$*

We say that the Phasing *protocol satisfies sender's privacy if for all PPT $\mathcal{A}$ the probability that $\mathcal{A}$ guesses b correctly is at most a negligible factor away from $1/2$.*

In the following, we will make use of the fact that the output of Phasing is indistinguishable from random. In particular this implies that it is indistinguishable from the T-KEM.Gen method (which is defined below), which allows to compose T-KEM and Phasing in Section 4.3.

Efficiency of Phasing For completeness, we also recall that in [40] the Phasing subroutine is implemented using a form of oblivious pseudorandom function (OPRF) which is in turn constructed efficiently thanks to oblivious transfer (OT) extension [19]. A trivial implementation of the Phasing subroutine using the OT induced OPRF would lead to the output of the sender U to have size n^2 (and the output of the receiver V to have size n). Combining the OPRF with clever data structures (e.g., Cuckoo hashing), the authors of Phasing managed to turn the size of the output of both parties to be $\mathcal{O}(n)$ ($|U| \leq 2.4n$ for reasonable sizes of n), which greatly improves in the overall complexity of the protocol.

4.2 Threshold Key Encapsulation Mechanism (T-KEM)

To construct a T-PSI protocol from the Phasing protocol family we introduce a new cryptographic tool which we choose to call *threshold key encapsulation mechanism* or T-KEM. A T-KEM is defined as follows:

Definition 10 (T-KEM). *A T-KEM scheme is defined by the three algorithms called* $(\mathsf{Gen}, \mathsf{Encap}, \mathsf{Decap})$ *with the following syntax:*

- *The generation algorithm*
$$\mathsf{Gen}(1^\lambda) \to u$$
on input a security parameter λ*, outputs random values* u *from* $\{0,1\}^\lambda$*.*
- *The key encapsulation algorithm*
$$\mathsf{Encap}(U, t) \to (k, H)$$
on input a set of strings $U = \{u | u \in \{0,1\}^\lambda\}$ *and a threshold* t*, outputs a random key* $k \in \{0,1\}^\lambda$ *and a "hint"* H*.*
- *The key decapsulation algorithm*
$$\mathsf{Decap}(V, H) \to k'$$
on input a set of strings $V = \{v | v \in \{0,1\}^\lambda\}$ *and a hint* H *outputs either* $k' = k$ *or some failure symbol* $\perp$*.*

The idea of a T-KEM is that the two parties both have a set of random numbers, from which they want to derive a common key if and only if the two sets overlap by a threshold number of t items. As an example, one can imagine a single set being generated at random and then stored on two separate unreliable storage units. Both parties retain one storage unit. Later on, the two users want to establish a secure channel between them by using the material on the storage units. As long as the storage is not "too faulty" i.e., the intersection between the two (now different) sets of keys is larger than some threshold t, then the two parties will be able to derive the same key. In the PSI setting, the keys will be generated using the Phasing scheme, which will guarantee the necessary overlap only if the initial sets have a large enough overlap.

We say that a T-KEM is *correct* if, for all V, U s.t. $|V \cap U| \geq t$ the following probability is at most negligible

$$\Pr[\mathsf{Decap}(V, H) \neq k | (k, H) \leftarrow \mathsf{Encap}(U, t)]$$

We define security of T-KEM with the following game between an adversary $\mathcal{A}$ and a challenger $\mathcal{C}$.

Definition 11 (T-KEM security). *We say that a T-KEM is* secure *if for all PPT adversary* $\mathcal{A}$*, the probability that* $\mathcal{A}$ *outputs* $b' = b$ *in the following game is at most a negligible factor away from* $1/2$*:*

1. *$\mathcal{A}$ chooses $t' < t$ and outputs t'*
2. *For all $i = 1..t$, $\mathcal{C}$ runs $u_i \leftarrow \mathsf{Gen}(1^\lambda)$*
3. *$\mathcal{C}$ defines $V = \{u_i\}_{i=1..t'}$ and $U = \{u_i\}_{i=1..t}$*
4. *$\mathcal{C}$ computes $(k_0, H) \leftarrow \mathsf{Encap}(V, t)$*
5. *$\mathcal{C}$ samples randomly $k_1 \leftarrow \{0,1\}^\lambda$ and $b \leftarrow \{0,1\}$*
6. *$\mathcal{C}$ outputs (U, H, k_b) to $\mathcal{A}$*
7. *$\mathcal{A}$ outputs b'*

4.3 T-PSI from Phasing and T-KEM

We now describe how to construct a T-PSI scheme using the introduced building blocks, namely the Phasing subroutine, a T-KEM, and any IND-CPA secure symmetric encryption scheme (E_k, D_k) for a key k.

Definition 12 (Threshold PSI functionality). *For any T-PSI protocol, the functionality is given as:*

$$\mathsf{TPSI}(A, B) \to (\perp, Z)$$

where

$$Z = \begin{cases} A \cap B & \text{if } |A \cap B| \geq t \\ \emptyset & \text{otherwise} \end{cases}$$

That is, the sender learns nothing and the receiver learns the intersection if and only if the size of the intersection is greater than t.

Now for a concrete protocol which, as shown later, produces the above functionality. The sender and receiver, with input sets A and B respectively, start by running the Phasing subroutine to calculate U and V. Then, the sender runs TPSI^S using U and the threshold t and sends the output to the receiver. The receiver runs TPSI^R using the received input from the sender as well as V to compute the final result Z.

Procedure $\mathsf{TPSI}^S(U, t)$:
- $(U_0, U_1) \leftarrow \mathsf{split}(\mathsf{U})$
- $(k, H) \leftarrow \mathsf{Encap}(U_0, t)$
- $C \leftarrow E_k(U_1)$
- return (C, H)

Proc. $\mathsf{TPSI}^R(V, R_V, C, H)$:
- $(V_0, V_1) \leftarrow \mathsf{split}(\mathsf{V})$
- $k' \leftarrow \mathsf{Decap}(V_0, H)$
- **if** $k' = \perp$ **then** :
  - return $\emptyset$
- **else** :
  - $U_1 \leftarrow D_{k'}(C)$
  - $Z \leftarrow \mathsf{Isect}(U_1, V_1, R_V)$
  - return Z

Elements in U and V are 2λ bits. The split subroutine enables creation of U_0 and V_0 as the sets containing all the top λ bits from all the masks in U, V and U_1, V_1 to be the sets containing all the bottom λ bits from all the masks in U, V. Splitting the masks in two is needed to avoid circular security issues between the T-KEM and the encryption scheme.

Definition 13 (Phasing/T-KEM T-PSI). *The protocol for Phasing/T-KEM T-PSI proceeds as follows:*

1. *The sender with input A and receiver with input B jointly run*

$$(U, (V, R_V)) \leftarrow \mathsf{Phasing}(A, B)$$

   The sender retains U and the receiver retains (V, R_V).
2. *The sender runs*

$$(C, H) \leftarrow \mathsf{TPSI}^S(U, t)$$

   and sends (C, H) to the receiver.
3. *The receiver concludes the protocol by computing*

$$Z \leftarrow \mathsf{TPSI}^R(V, R_V, C, H)$$

   and outputting Z.

Security of T-PSI As highlighted before, the construction should respect the functionality described in Definition 12. Thus, Theorem 2 captures the security goal of our protocol.

Theorem 2 (Phasing/T-KEM security). *The protocol specified in Definition 13 securely implements the privacy-preserving threshold set intersection functionality (Definition 12) in the presence of passive adversaries*

Proof. The security of the protocol follows from the security of the building blocks. It is trivial to argue security against a passively corrupt sender, since the view of the sender in the T-PSI protocol is exactly the same as in the original Phasing protocol (the sender does not receive any extra messages).

We now argue security against a passively corrupt receiver. To do so, we need to construct a simulator that, having access to the desired input/output of the receiver (e.g., B and Z and in particular not to the sender's input A), constructs a simulated view which is computationally indistinguishable from the view of the receiver in the real protocol. The simulator is constructed with two main cases – when an intersection larger than t is known, and when it is not.

With known threshold: If $|A \cap B| \geq t$, we simply run the simulator of the Phasing protocol. In particular, the simulator runs $\mathcal{S}_R^{\text{PHG}}(B, A \cap B)$ to compute sets of masks (U, V). Now the simulator parses $U = (U_0, U_1)$ and $V = (V_0, V_1)$, computes $(k, H) \leftarrow \mathsf{Encap}(U_0, t)$, encrypts $C = E_k(U_1)$ and adds (C, H) to the simulated view. Any adversary that can distinguish between this view and the view in the real protocol can be used to break the security of the Phasing subroutine using the following reduction: the reduction chooses an arbitrary set B' such that $A \cap B = A \cap B'$ and outputs A, B' to the challenger and receives (U, V). Now the reduction completes the view of the protocol by recomputing (k, H, C) and adding (C, H) to the view, which is passed to the adversary against the protocol simulator. By the construction of Definition 9, when $b = 0$ this corresponds to the real view of the protocol execution and when $b = 1$ this corresponds to the simulated run, hence the reduction breaks Definition 9 with the same advantage as the adversary for the overall protocol.

Without known threshold: The more interesting case is of course when $|A \cap B| < t$. In this case the simulator runs the Phasing simulator $\mathcal{S}_R^{\text{PHG}}(B, \emptyset)$ and receives the masks U^*, V^* such that $U^* \cap V^* = \emptyset$. Next the simulator parses $U^* = (U_0^*, U_1^*)$ and $V^* = (V_0^*, V_1^*)$, computes $(k^*, H^*) \leftarrow \mathsf{Encap}(U_0^*, t)$, encrypts $C^* = E_{k^*}(U_1^*)$ and adds (C^*, H^*) to the simulated view. Note that the distribution of the simulated view is different from the distribution in the real protocol. In particular in the real protocol U, V are such that $|U \cap V| = |A \cap B| < t$ while in the simulation $|U^* \cap V^*| = 0$. We argue that the views are computationally indistinguishable anyway. We do so by a sequence of hybrid games, where hybrid 0 is identical to the real protocol. In the first hybrid we replace k with k^* (the simulated key), and therefore we replace $C = E_k(U_1)$ with $C' = E_{k^*}(U_1)$. Any adversary that can distinguish between these two hybrids can be used to break the security of the underlying T-KEM scheme (in the reduction the adversary here has only $|A \cap B| < t$ of the necessary masks). In the second hybrid we replace the encrypted masks U_1 with the simulated masks U_1^*. That is, in the view we replace $C' = E_{k^*}(U_1)$ with $C^* = E_{k^*}(U_1^*)$. Any adversary that distinguishes between these two hybrids can be used to break the IND-CPA security of the encryption scheme (note that the reduction here does not need to know the key). Since the second hybrid is identical to the simulated view, this concludes the proof.

□

5 Implementing T-KEM

This section describes an implementation of the T-KEM algorithms that satisfies the security requirements. Our techniques are based on polynomial interpolation, and can be seen as an extension of the well known Shamir's secret sharing [46] scheme.

Encapsulation Given a threshold t, we implement $\mathsf{Encap}(B, t)$ on an input set B with $|B| = n > t$ in the following way: first we parse every element in B as $(x_i^B, y_i^B) \in \mathbb{F} \times \mathbb{F}$ where $\mathbb{F}$ is an appropriately large (exponential) finite field. Then we define a polynomial p of degree $n-1$ from the n points known to B as $p(x_i^B) = y_i^B$ and the output key as $k = p(0)$

The hint is composed of $n - t$ additional points (x_i^h, y_i^h) with $y_i^h = p(x_i^h)$ as well as a hash of the key $ck = R(k)$ (where R is modeled as a random oracle) which allows to detect correct decapsulation:

$$H = (ck = R(k), \{(x_i^h, y_i^h)\}_{i \in \{1,\ldots,n-t\}})$$

Decapsulation The decapsulation process $\mathsf{Decap}(H, A)$ works as follows: first parse A as $(x_i^A, y_i^A) \in \mathbb{F} \times \mathbb{F}$; then, for every subset $A' \subseteq A$ with $|A'| = t$, define a polynomial p' using the points in H and in A', then compute a candidate key $k' = p'(0)$ and check if $R(k') = ck$. If yes, then output $k = k'$, otherwise proceed with the next subset A'.

The correctness follows from the basic fact that if two polynomials p, p' of degree $n-1$ agree on n points, then $p = p'$ [46].

To show security as per Definition 11, we make the following argument. Security follows from the fact that (in the random oracle model) ck can only be used to verify if $k' = k$ but otherwise leak no information about k. Consider the worst case, in which the adversary has $t' = t - 1$ correct points. Then even given the $n - t$ hints the adversary only has $n-1$ points on a $n-1$ degree polynomial, which means that the best chance of guessing k is $|\mathbb{F}|^{-1}$. Since ck can only be used to verify guesses, then the only way of distinguishing the real k_0 from the random key k_1 (in the security game of Definition 11) is to invert the random oracle, and the probability that an adversary can do this by querying the oracle $q = \mathsf{poly}(\lambda)$ times is bounded by $q/|\mathbb{F}|$ which is negligible if the size of $\mathbb{F}$ is exponential in the security parameter.

In terms of efficiency, the main bottlenecks in the solution described above is to construct the hint during encapsulation, and how to reconstruct the key k from the receiver's set and the hints during decapsulation. How to accomplish encapsulation

and decapsulation in the context of ridesharing is detailed in Appendix 1. Encapsulation is rather straight-forward with a running time of $\mathcal{O}(n^3)$. Interestingly, for the corner case when any overlap of the input sets are sequential, as is the case for ridesharing applications, decapsulation can be optimized to give an asymptotic running time of $\mathcal{O}(n^2)$.

6 Experiments

This section details experiments to assert both the effectiveness and efficiency of our two-faceted approach. First, we present a study of taxi trips in New York City. This study compares how many trips the two privacy-preserving approaches find in comparison to a plaintext implementation. Secondly, the efficiency of the two approaches are evaluated, and compared to a set intersection protocol implemented in a generic SMC framework using garbled circuits.

6.1 Study of Taxi rides

To determine the effectiveness of the privacy-preserving approaches, we have evaluated our approach on real-world data of taxi rides in New York City, as publicly provided by the New York City Taxi and Limousine Commission (TLC) [51]. We have used this data to find typical movements in a dense city, assuming the user would also be willing to travel using other means than by taxi to go from their origin to their destination. Given any two taxi rides, we assign one to be party A, who is willing to deviate from their route, and the other party B who sticks to their shortest route. TLC reveals only the origin and destination of the users. To calculate the route, we have used the open source software Routino [44], utilizing street data from OpenStreetMap [37].[2]

The taxi dataset is very large, but there are surprisingly few rides that happen at the same time such that the users could have shared a cab in practice. We conjecture that the paths computed from the taxi trips are realistic movement patterns, but that the times the rides take place are likely widely different from users prone to use ride-sharing services in order to e.g. commute. Even though taxi trips in New York City do not generalize to mobility patterns in general (it is even likely that users of the yellow non-bookable cabs differ from users of the green pre-booked cabs within New York City alone), this data at least give some indicative intuition for ridesharing

[2] Map data copyrighted OpenStreetMap contributors and available from http://www.openstreetmap.org

in a dense city. Thus, in this study we opt to not look at the time of day at all, and instead view each file in the dataset as a time-frame in which ridesharing is feasible from a timing perspective. Files are grouped month by month, which means we effectively get a relative measure between the different patterns which is roughly averaged over time of day, and weekday, and so forth. Though the magnitude of ridesharing opportunities calculated in the study is an over-approximation, the goal is not to find how many ridesharing opportunities exist, but which pattern detects the most ridesharing opportunities.

As the TLC dataset is vast, the "bruteforce" algorithm (outlined in Section 3) is the bottleneck in the comparison. To make the comparison feasible, only 1000 trips from every even month of 2015 for the green cab dataset was used. The dataset was considered as asymmetric, such that both parties can assume the role of A or B, giving roughly one million possible ridesharing opportunities every month and a total of 6 million for all months.

The number of ridesharing opportunities found when using intersection-based matching, endpoint-based matching, and when brute-forcing all points according to the "bruteforce" algorithm outlined were counted. The bruteforce algorithm captures all rides caught by the model. We assume that for this dataset, users are willing to deviate most the start and end of their trajectory, and not at all in the middle. The intuition is that the user familiar to the areas close to home and/or work, but less comfortable with improvising further along the route. A second-degree polynomial was used to implement the deviation function Δ, as shown in Equation 3.

$$\Delta_T(i) = 4i^2 \frac{r_0}{|T|^2} - 4i\frac{r_0}{|T|} + r_0 \tag{3}$$

In Equation 3, r_0 is how much the user is willing to deviate at the endpoints, and i is the index of a vertex such that $\forall i : v_i \in T$. This gives a "happy-smiley-face" curve with root at $|T|/2$ with $y = r_0$ at $x = 0$ and $x = |T|$. Note that the indices of the positions rather than the positions themselves does lead to some imprecision. But Equation 3 gives an intuitive estimation with the right sign on the derivate in almost all points.

Table 1 illustrates the effectiveness of the two different approaches on this particular dataset. In the table, t says what percentage of the trip must be shared for ridesharing to be deemed feasible, and r_0 is the deviation allowed at the start and endpoints (with $\Delta_T(\cdot)$ as per Equation 3). It shows the number of found ridesharing opportunities by both approaches in PrivatePool (PP), intersection-based matching (IS) and endpoint-based matching (EP) compared to capturing all opportunities when applying the entire model. The results show that effectiveness is heavily re-

liant on the parameters, ranging from over 92% to 14% when utilizing both approaches, 11% to 0.3% for endpoint matching, and 92% to 2.5% when using only intersection. PrivatePool thus captures the vast majority as covered by the model, but which privacy-preserving approach to use depends on the needs of the user. Even though the highest results were achieved with intersection-based matching, the experiments confirm that endpoint matching is sometimes preferable to the intersection-based approach.

Table 1. Effectiveness of privacy-preserving approaches (in percent)

t	$r_0 = 500$			$r_0 = 1000$			$r_0 = 2000$		
	PP	IS	EP	PP	IS	EP	PP	IS	EP
20%	92.6	92.29	0.31	61.3	59.98	1.31	36.91	31.92	4.99
50%	75.57	74.64	0.93	45.73	42.3	3.43	25.26	16.01	9.25
80%	28.76	26.24	2.52	15.06	9.26	5.8	13.57	2.48	11.09

6.2 Benchmarks

To benchmark the different approaches, the two patterns as outlined in Section 2 were investigated, again using OSM and Routino. The shortest path for several likely routes from intra-city rides to trips over 1000 km, were used to determine the size of the input set to use for the benchmarks. As seen in Table 2, the standard deviation is very large, meaning that the road sections are of very varying size. Further, it is visible that for shorter trips, the road segments are shorter, likely because a highway contains long road sections.

The implementations measure only the CPU time, and leaves out I/O operations entirely. Benchmarks were done on a single machine with 16GB RAM and an Intel i7-4790 CPU at 3.60GHz.

Implementation details For the different approaches, separate implementations were used. For endpoint matching, the implementation is built on top of the original python code for proximity testing by Hallgren et al., whereas for the T-PSI a new C++ prototype for T-KEM was developed for integration into the work by Pinkas et al.[40], but should also be compatible with the most efficient Phasing subroutine from the work of Kolesnikov et al. [26]. For a comparison to generic circuit-based approaches, a Java implementation (called FastGC) for PSI by Huang et al. [17] was

Table 2. Road segment lengths from OpenStreetMap data

Number of sections	Distance(m)	Average (m)	Standard deviation (m)
230	12013	52.46	81.87
123	19801	162.30	160.13
626	143165	229.06	217.20
1856	610581	329.15	285.69
2420	623177	257.62	231.94
2142	735975	343.75	305.62
3440	931475	270.86	308.35
2868	974107	339.77	299.89

used. For each implementation there are different perks and drawbacks, and in many ways the comparison is unfair (e.g. by programming language). However, regarding performance the results firmly position the different solutions asymptotically relative to each other. Exactly how they are configured is detailed in the following.

For the end-point matching, we are able to achieve very good results for coarse grained precision. The implementation is using 1024 bit keys for the Paillier cryptosystem, and a "radius value" of 4, with an imprecision of 500 meters. This means that proximity is checked in a 2 km range, where however both false negatives and false positives are possible up to 500 meters.

As the data received from OpenStreetMap does not correspond to equal-sized chunks (road sections have different lengths), the data needs to be processed to be a good fit for the two PSI solutions. For an application that needs to be precise in the intersection threshold t, the road sections would need to be split into sections as long as the distance unit of OpenStreetMap. This would greatly increase the size of the input set for the PSI solutions. For the scope of this work, these smaller input sizes suffices to evaluate the efficiency relative to the other approaches, and as will be seen below the used inputs are enough to see roughly for which size of the input the different solutions are useful. For our benchmarks, the input set corresponds to using the start coordinate of each road section. Though this much imprecision often would cause the intersection to be either half or twice as long as the intended threshold, plaintext preprocessing can be used to attune precision and mitigate this issue once parameters are fixed for a certain application.

The T-KEM implementation uses $\lambda = 128$, utilizing the Intel Intrinsic [12] instruction set for operations in a field of size 2^{128} and t was set to 80% of the total set size. On the sender side, parallelization is trivial, and the sender thus computes hints in parallel using OpenMP [36]. Though the benchmarks shown do not include the

Phasing subroutine, the full original protocol needs under 0.3 seconds for 4096-bit set sizes [40], rendering it almost negligible in the comparison.

The GC solution for PSI uses 80-bit security and 32-bit strings for the input elements. For the scope of this work it was chosen to not include the threshold computation. The addition of the extra comparison should yield significantly slower results for the generic solution. However, the asymptotic result shown in the following section should not change – for large sets the generic solution should outperform our threshold version. To determine how much larger the set needs to be for a threshold GC solution to outperform our T-PSI protocol is left for future work.

Notably, there are more efficient implementations for generic SMC which should be used for further investigations. The recent work by Pinkas et al. [41] show results which are significantly better than those of Huang et al. With their setup for 128-bit security on a LAN network, a set intersection between 256 elements was performed in 0.304 seconds and 4096 elements in 1.647 seconds, which means that they may outperform our T-PSI solution also for very small input sets.

Results Benchmarks using the three prototypes show that all are feasible enforcements in some situations, see Figure 7. Table 3 shows the average time taken to check for ridesharing opportunities using endpoint-matching, T-PSI, and FastGC set intersection. Experiments were run 25 times to minimize noise, with a less than 10% standard deviation.

Table 3. Benchmarks (seconds)

Number of	End-point	T-PSI			FastGC
sections	match (r=4)	Sender	Receiver	Total	PSI
32	0.352	0.014	0.008	0.022	2.485
64	0.367	0.046	0.031	0.077	2.915
128	0.361	0.174	0.187	0.362	3.918
256	0.363	0.695	1.388	2.083	5.593
512	0.372	2.778	10.79	13.57	8.665
1024	0.354	11.12	85.67	96.78	15.100
2048	0.353	44.34	683.8	728.1	30.263
4096	0.371	176.8	5450	5627	62.154

The end-point matching performs independently of the size of the set, however suffers greatly with improved precision. At $r = 25$ (80 meter precision) the implementation finishes in 9.2 seconds, and at $r = 100$ (20 meter precision) in 124.43

seconds. At this stage, the derivate tapers off (for small r the increase in running time is close to quadratic, for large r it's close to linear), but large values of r are out of reach for practical applications with the chosen approach.

For the intersection-based solutions, the FastGC implementation is only preferable over T-PSI for very large sets, and though it outperforms our threshold variant for paths longer than 500 items as per Table 3, both implementations are arguably too slow to be used in practice at this stage. Thus, the T-PSI implementation is preferable over FastGC for any application where the user cannot tolerate large delays, such as consumer-facing applications. Overall, the benchmarks are very encouraging results for using SMC techniques. Though indeed some configurations are very impractical, for most settings there are alternatives that yield satisfying results. The endpoint matching and intersection approaches are both very efficient for low-precision applications in the general case, and T-PSI can be used with full precision for shorter trips. Both could be used in many applications, and for any consumer-facing such as ridesharing they outperform generic techniques by orders of magnitude.

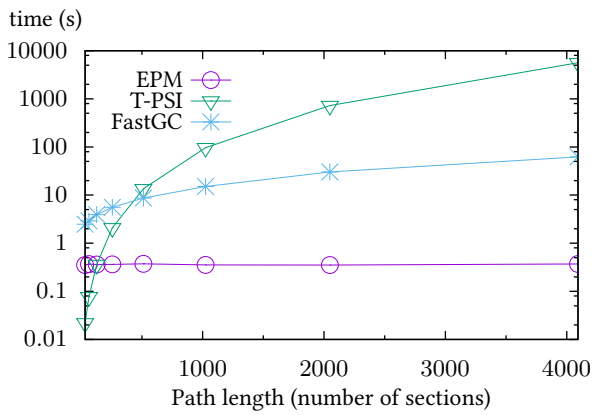

Fig. 7. Visualization of the benchmark results

Much important work within SMC is of a more theoretical nature, and is thus not able to show that SMC is applicable to at least some real-world scenarios, e.g. using T-PSI for short trips. These experiments show how to use SMC as a part of a product in the emerging area of digital transportation services. Indeed, though all evaluated approaches need to use a degraded precision in the general case, the achieved running time is well within reach of practical applications with acceptable precision for many settings.

7 Related work

As discussed in recent surveys [28, 50], location privacy is an increasingly important topic. Relating to the privacy of a user's trajectory, much prior research relates to re-identification [8, 4], whereas our work focuses on minimizing disclosure of location data. The focus of hiding the identity of a user stems from the common fact that many existing parties has a trace of a user's whereabouts and wants to publish statistical facts about crowds. E.g. a provider of public transport which uses electronic tickets often knows where a user enters and exits a vehicle, and might want to contribute anonymized data for urban planning [23]. Instead, this work focuses on the case where a service provider, who anyways needs to know the identity of their customers, wants to minimize the data disclosure to e.g. reduce the risk of security breaches or to conform to emerging regulations, such as the data protection regulation by the European Union [52].

There is extensive literature on the problem of how to test for the proximity of two points as provided by two different users, without revealing more than the proximity result [57, 48, 47, 7, 32, 35, 45, 15]. However, most work considers a single proximity result in isolation. There are some exceptions, such as the work by Hallgren et al. [16] to account for moving parties. In the spirit of Hallgren et al., we utilize multiplication of two proximity results to check whether both are positive. There are several ad-hoc solutions which are not amenable to this approach [48, 47, 7, 32]. Further work uses additive homomorphic encryption for location proximity [57, 35, 45, 15].

The first problem considered for SMC was the millionaires problem, where two parties wanted to find which was the greater of two integers [56], from which equality testing [5] and subsequently set intersection are natural steps [6]. Set intersection is an important primitive needed to build many larger applications, as discussed in several works [17, 40, 39]. Set intersection has been implemented both using homomorphic encryption [18, 22], garbled circuits [17], and ad-hoc solutions [40, 39, 26]. Of these, the ad-hoc solutions perform better than the other techniques for small input sizes, with garbled circuit-based solutions being asymptotically slower for large input sets [40].

8 Conclusions

We have presented an approach to specifying and enforcing privacy in ridesharing applications. Our investigation of ridesharing patterns has elucidated the benefits

of a two-fold approach: (i) ride matching based on the proximity of start- and endpoints of the rides and (ii) ride matching based on the overlap of the ride trajectories. We have therefore developed privacy-preserving mechanisms for (i) start/end point matching and (ii) private trajectory matching. For the former, we have built on recent work on location proximity based on homomorphic encryption. For the latter, we have designed a novel protocol for threshold private set intersection (dubbed T-PSI), based on Shamir's secret sharing scheme.

We have evaluated the effectiveness of our approach on the real-world data from the New York City Taxi and Limousine Commission, confirming that the endpoint-based and intersection-based mechanisms are both useful.

We have prototyped and benchmarked these mechanisms and contrasted them with a general-purpose approach based on garbled circuits. The evaluation shows that for any application that requires termination under 10 seconds, both of our mechanisms outperform the generic garbled-circuit approach. The benchmarks also indicate that precision can be traded to achieve running time in under a second using either homomorphic encryption or the T-PSI protocol.

Future Work Our results open up for promising future work tracks. On the theoretical side, we plan to investigate multi-key protocols for ridesharing. If successful, this will allow us to boost the scalability of the approach by significantly simplifying the key distribution phase. On the practical side, we plan to build a fully-fledged ridesharing app, drawing on the infrastructure developed in our prototype.

Moreover, we're keen to conduct further investigation to find more efficient schemes for T-PSI. The simple scheme outlined in the following could outperform our solution based on T-KEM, where Alice has a trajectory T^A of size m and Bob a trajectory T^B of size n, for any given a threshold t.

Note that this solution only works when the input data is sorted as in our case for ridesharing. We need that if $\exists i, j : T_i^A = T_j^B \wedge T_{i+t}^A = T_{j+t}^B$ then it also holds that $\forall u \in \{i, \ldots, i+t\}, v \in \{j, \ldots, j+t\} : T_u^A = T_v^B . \forall u \in \{0, \ldots, t\} : T_{i+u}^A = T_{j+u}^B$. The construction proceeds as follows:

1. Alice prepares a set on the following form,

$$(T_1^A, T_t^A), (T_2^A, T_{t+1)}^A, \ldots, (T_{m-t+1}^A, T_m^A)$$

2. Bob similarly prepares a set on the form

$$(T_1^B, T_t^B), (T_2^B, T_{t+1)}^B, \ldots, (T_{n-t+1}^B, T_n^B)$$

3. The parties run a standard PSI protocol on these sets.

Each element is a pair in the original set denoting a segment of length t. Thus, if the PSI returns at least one element, the parties may rideshare along the segment represented by this element. It's easy to see that shared segments of length less than t are not disclosed. All that remains is to find the maximum segment to use for ridesharing, but this can be done in the plain by the receiver of the PSI by taking the earliest and latest coordinate in any returned pair.

Acknowledgments Thanks are due to Peter Druschel and Rijurekha Sen for inspiring discussions on the scenario of private ridesharing. This work was partly funded by the European Community under the ProSecuToR project, COST Action IC1306, Danish Independent Research Council and the Swedish research agency VR.

References

1. C. Bessette, "Does Uber Even Deserve Our Trust?" http://www.forbes.com/sites/chanellebessette/2014/11/25/does-uber-even-deserve-our-trust/, Nov. 2014.
2. "BlaBlaCar - Trusted carpooling," https://www.blablacar.com/.
3. D. Chaum, C. Crépeau, and I. Damgård, "Multiparty unconditionally secure protocols (extended abstract)," in *Proceedings of the 20th Annual ACM Symposium on Theory of Computing, May 2-4, 1988, Chicago, Illinois, USA*, J. Simon, Ed. ACM, 1988, pp. 11–19. [Online]. Available: http://doi.acm.org/10.1145/62212.62214
4. R. Chen, B. C. M. Fung, and B. C. Desai, "Differentially private trajectory data publication," *CoRR*, vol. abs/1112.2020, 2011. [Online]. Available: http://arxiv.org/abs/1112.2020
5. R. Fagin, M. Naor, and P. Winkler, "Comparing information without leaking it," *Commun. ACM*, vol. 39, no. 5, pp. 77–85, 1996. [Online]. Available: http://doi.acm.org/10.1145/229459.229469
6. M. J. Freedman, K. Nissim, and B. Pinkas, "Efficient private matching and set intersection," in *Advances in Cryptology - EUROCRYPT 2004, International Conference on the Theory and Applications of Cryptographic Techniques, Interlaken, Switzerland, May 2-6, 2004, Proceedings*, ser. Lecture Notes in Computer Science, C. Cachin and J. Camenisch, Eds., vol. 3027. Springer, 2004, pp. 1–19. [Online]. Available: \url{http://dx.doi.org/10.1007/978-3-540-24676-3_1}
7. D. Freni, C. R. Vicente, S. Mascetti, C. Bettini, and C. S. Jensen, "Preserving location and absence privacy in geo-social networks," in *Proceedings of the 19th ACM Conference on Information and Knowledge Management, CIKM 2010, Toronto, Ontario, Canada, October 26-30, 2010*, J. Huang, N. Koudas, G. J. F. Jones, X. Wu, K. Collins-Thompson, and A. An, Eds. ACM, 2010, pp. 309–318. [Online]. Available: http://doi.acm.org/10.1145/1871437.1871480
8. G. Ghinita, "Private queries and trajectory anonymization: a dual perspective on location privacy," *Trans. Data Privacy*, vol. 2, no. 1, pp. 3–19, 2009. [Online]. Available: http://www.tdp.cat/issues/abs.a018a09.php

9. A. A. Ghorbani, V. Torra, H. Hisil, A. Miri, A. Koltuksuz, J. Zhang, M. Sensoy, J. García-Alfaro, and I. Zincir, Eds., *13th Annual Conference on Privacy, Security and Trust, PST 2015, Izmir, Turkey, July 21-23, 2015*. IEEE, 2015. [Online]. Available: http://ieeexplore.ieee.org/xpl/mostRecentIssue.jsp?punumber=7193739
10. O. Goldreich, *The Foundations of Cryptography - Volume 2, Basic Applications*. Cambridge University Press, 2004.
11. O. Goldreich, S. Micali, and A. Wigderson, "How to play any mental game or A completeness theorem for protocols with honest majority," in *Proceedings of the 19th Annual ACM Symposium on Theory of Computing, 1987, New York, New York, USA*, A. V. Aho, Ed. ACM, 1987, pp. 218–229. [Online]. Available: http://doi.acm.org/10.1145/28395.28420
12. S. Gueron and M. E. Kounavis, "Intel® carry-less multiplication instruction and its usage for computing the gcm mode," https://software.intel.com/sites/default/files/managed/72/cc/clmul-wp-rev-2.02-2014-04-20.pdf, Apr. 2014.
13. S. Halevi, Y. Lindell, and B. Pinkas, "Secure computation on the web: Computing without simultaneous interaction," in *Advances in Cryptology - CRYPTO 2011 - 31st Annual Cryptology Conference, Santa Barbara, CA, USA, August 14-18, 2011. Proceedings*, ser. Lecture Notes in Computer Science, P. Rogaway, Ed., vol. 6841. Springer, 2011, pp. 132–150. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-22792-9_8
14. P. A. Hallgren, M. Ochoa, and A. Sabelfeld, "Bettertimes - privacy-assured outsourced multiplications for additively homomorphic encryption on finite fields," in *Provable Security - 9th International Conference, ProvSec 2015, Kanazawa, Japan, November 24-26, 2015, Proceedings*, ser. Lecture Notes in Computer Science, M. H. Au and A. Miyaji, Eds., vol. 9451. Springer, 2015, pp. 291–309. [Online]. Available: \url{http://dx.doi.org/10.1007/978-3-319-26059-4_16}
15. ——, "Innercircle: A parallelizable decentralized privacy-preserving location proximity protocol," in *13th Annual Conference on Privacy, Security and Trust, PST 2015, Izmir, Turkey, July 21-23, 2015*, A. A. Ghorbani, V. Torra, H. Hisil, A. Miri, A. Koltuksuz, J. Zhang, M. Sensoy, J. García-Alfaro, and I. Zincir, Eds. IEEE, 2015, pp. 1–6. [Online]. Available: \url{http://dx.doi.org/10.1109/PST.2015.7232947}
16. ——, "Maxpace: Speed-constrained location queries," in *2016 IEEE Conference on Communications and Network Security, CNS 2016, Philadelphia, PA, USA, October 17-19, 2016*, 2016.
17. Y. Huang, D. Evans, and J. Katz, "Private set intersection: Are garbled circuits better than custom protocols?" in *19th Annual Network and Distributed System Security Symposium, NDSS 2012, San Diego, California, USA, February 5-8, 2012*. The Internet Society, 2012. [Online]. Available: http://www.internetsociety.org/private-set-intersection-are-garbled-circuits-better-custom-protocols
18. B. A. Huberman, M. K. Franklin, and T. Hogg, "Enhancing privacy and trust in electronic communities," in *EC*, 1999, pp. 78–86. [Online]. Available: http://doi.acm.org/10.1145/336992.337012

19. Y. Ishai, J. Kilian, K. Nissim, and E. Petrank, "Extending oblivious transfers efficiently," in *Advances in Cryptology - CRYPTO 2003, 23rd Annual International Cryptology Conference, Santa Barbara, California, USA, August 17-21, 2003, Proceedings*, ser. Lecture Notes in Computer Science, D. Boneh, Ed., vol. 2729. Springer, 2003, pp. 145–161. [Online]. Available: http://dx.doi.org/10.1007/978-3-540-45146-4_9
20. J. Jung and T. Holz, Eds., *24th USENIX Security Symposium, USENIX Security 15, Washington, D.C., USA, August 12-14, 2015*. USENIX Association, 2015. [Online]. Available: https://www.usenix.org/conference/usenixsecurity15
21. F. Kerschbaum, "Adapting privacy-preserving computation to the service provider model," in *Proceedings of the 12th IEEE International Conference on Computational Science and Engineering, CSE 2009, Vancouver, BC, Canada, August 29-31, 2009*. IEEE Computer Society, 2009, pp. 34–41. [Online]. Available: http://dx.doi.org/10.1109/CSE.2009.261
22. ——, "Outsourced private set intersection using homomorphic encryption," in *7th ACM Symposium on Information, Compuer and Communications Security, ASIACCS '12, Seoul, Korea, May 2-4, 2012*, H. Y. Youm and Y. Won, Eds. ACM, 2012, pp. 85–86. [Online]. Available: http://doi.acm.org/10.1145/2414456.2414506
23. H. Kikuchi and K. Takahashi, "Zipf distribution model for quantifying risk of re-identification from trajectory data," in *13th Annual Conference on Privacy, Security and Trust, PST 2015, Izmir, Turkey, July 21-23, 2015*, A. A. Ghorbani, V. Torra, H. Hisil, A. Miri, A. Koltuksuz, J. Zhang, M. Sensoy, J. García-Alfaro, and I. Zincir, Eds. IEEE, 2015, pp. 14–21. [Online]. Available: http://dx.doi.org/10.1109/PST.2015.7232949
24. L. Kissner and D. X. Song, "Private and threshold set-intersection," http://www.dtic.mil/dtic/tr/fulltext/u2/a461119.pdf, 2004.
25. ——, "Privacy-preserving set operations," in *Advances in Cryptology - CRYPTO 2005: 25th Annual International Cryptology Conference, Santa Barbara, California, USA, August 14-18, 2005, Proceedings*, ser. Lecture Notes in Computer Science, V. Shoup, Ed., vol. 3621. Springer, 2005, pp. 241–257. [Online]. Available: http://dx.doi.org/10.1007/11535218_15
26. 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, Vienna, Austria, October 24-28, 2016*, E. R. Weippl, S. Katzenbeisser, C. Kruegel, A. C. Myers, and S. Halevi, Eds. ACM, 2016, pp. 818–829. [Online]. Available: http://doi.acm.org/10.1145/2976749.2978381
27. V. Kolesnikov, A. Sadeghi, and T. Schneider, "From dust to dawn: Practically efficient two-party secure function evaluation protocols and their modular design," *IACR Cryptology ePrint Archive*, vol. 2010, p. 79, 2010. [Online]. Available: http://eprint.iacr.org/2010/079
28. J. Krumm, "A survey of computational location privacy," *Personal and Ubiquitous Computing*, vol. 13, no. 6, pp. 391–399, 2009.
29. J. Krumm and E. Horvitz, "LOCADIO: inferring motion and location from wi-fi signal strengths," in *1st Annual International Conference on Mobile and Ubiquitous Systems (MobiQuitous 2004), Networking and Services, 22-25 August 2004, Cambridge,*

MA, USA. IEEE Computer Society, 2004, pp. 4–13. [Online]. Available: http://dx.doi.org/10.1109/MOBIQ.2004.1331705

30. Y. Lindell and B. Pinkas, "Secure multiparty computation for privacy-preserving data mining," *IACR Cryptology ePrint Archive*, vol. 2008, p. 197, 2008. [Online]. Available: http://eprint.iacr.org/2008/197
31. "Lyft," https://www.lyft.com/.
32. S. Mascetti, D. Freni, C. Bettini, X. S. Wang, and S. Jajodia, "Privacy in geo-social networks: proximity notification with untrusted service providers and curious buddies," *VLDB J.*, vol. 20, no. 4, pp. 541–566, 2011. [Online]. Available: \url{http://dx.doi.org/10.1007/s00778-010-0213-7}
33. Y. Michalevsky, A. Schulman, G. A. Veerapandian, D. Boneh, and G. Nakibly, "Powerspy: Location tracking using mobile device power analysis," in *24th USENIX Security Symposium, USENIX Security 15, Washington, D.C., USA, August 12-14, 2015.*, J. Jung and T. Holz, Eds. USENIX Association, 2015, pp. 785–800. [Online]. Available: https://www.usenix.org/conference/usenixsecurity15/technical-sessions/presentation/michalevsky
34. K. Muthukrishnan, B. van der Zwaag, and P. J. M. Havinga, "Inferring motion and location using WLAN RSSI," in *Mobile Entity Localization and Tracking in GPS-less Environnments, Second International Workshop, MELT 2009, Orlando, FL, USA, September 30, 2009. Proceedings*, ser. Lecture Notes in Computer Science, R. Fuller and X. D. Koutsoukos, Eds., vol. 5801. Springer, 2009, pp. 163–182. [Online]. Available: http://dx.doi.org/10.1007/978-3-642-04385-7_12
35. A. Narayanan, N. Thiagarajan, M. Lakhani, M. Hamburg, and D. Boneh, "Location privacy via private proximity testing," in *Proceedings of the Network and Distributed System Security Symposium, NDSS 2011, San Diego, California, USA, 6th February - 9th February 2011*. The Internet Society, 2011. [Online]. Available: http://www.isoc.org/isoc/conferences/ndss/11/pdf/1_3.pdf
36. OpenMP Architecture Review Board, "OpenMP application program interface version 4.0," May 2013. [Online]. Available: http://www.openmp.org/wp-content/uploads/OpenMP4.0.0.pdf
37. "Openstreetmap," http://www.openstreetmap.org/.
38. W. R. Ouyang, A. K. Wong, and C. A. Lea, "Received signal strength-based wireless localization via semidefinite programming: Noncooperative and cooperative schemes," *IEEE Trans. Vehicular Technology*, vol. 59, no. 3, pp. 1307–1318, 2010. [Online]. Available: http://dx.doi.org/10.1109/TVT.2010.2040096
39. B. Pinkas, T. Schneider, G. Segev, and M. Zohner, "Phasing: Private set intersection using permutation-based hashing," in *24th USENIX Security Symposium, USENIX Security 15, Washington, D.C., USA, August 12-14, 2015.*, J. Jung and T. Holz, Eds. USENIX Association, 2015, pp. 515–530. [Online]. Available: https://www.usenix.org/conference/usenixsecurity15/technical-sessions/presentation/pinkas
40. B. Pinkas, T. Schneider, and M. Zohner, "Faster private set intersection based on OT extension," in *Proceedings of the 23rd USENIX Security Symposium, San Diego,*

CA, USA, August 20-22, 2014., K. Fu and J. Jung, Eds. USENIX Association, 2014, pp. 797–812. [Online]. Available: https://www.usenix.org/conference/usenixsecurity14/technical-sessions/presentation/pinkas

41. ——, "Scalable private set intersection based on OT extension," *IACR Cryptology ePrint Archive*, vol. 2016, p. 930, 2016. [Online]. Available: http://eprint.iacr.org/2016/930
42. I. Polakis, G. Argyros, T. Petsios, S. Sivakorn, and A. D. Keromytis, "Where's wally?: Precise user discovery attacks in location proximity services," in *Proceedings of the 22nd ACM SIGSAC Conference on Computer and Communications Security, Denver, CO, USA, October 12-6, 2015*, I. Ray, N. Li, and C. Kruegel, Eds. ACM, 2015, pp. 817–828. [Online]. Available: http://doi.acm.org/10.1145/2810103.2813605
43. P. Rindal and M. Rosulek, "Improved private set intersection against malicious adversaries," in *Advances in Cryptology - EUROCRYPT 2017 - 36th Annual International Conference on the Theory and Applications of Cryptographic Techniques, Paris, France, April 30 - May 4, 2017, Proceedings, Part I*, ser. Lecture Notes in Computer Science, J. Coron and J. B. Nielsen, Eds., vol. 10210, 2017, pp. 235–259. [Online]. Available: http://dx.doi.org/10.1007/978-3-319-56620-7_9
44. "Routino," http://www.routino.org/.
45. J. Sedenka and P. Gasti, "Privacy-preserving distance computation and proximity testing on earth, done right," in *9th ACM Symposium on Information, Computer and Communications Security, ASIA CCS '14, Kyoto, Japan - June 03 - 06, 2014*, S. Moriai, T. Jaeger, and K. Sakurai, Eds. ACM, 2014, pp. 99–110. [Online]. Available: http://doi.acm.org/10.1145/2590296.2590307
46. A. Shamir, "How to share a secret," *Commun. ACM*, vol. 22, no. 11, pp. 612–613, 1979. [Online]. Available: http://doi.acm.org/10.1145/359168.359176
47. L. Siksnys, J. R. Thomsen, S. Saltenis, and M. L. Yiu, "Private and flexible proximity detection in mobile social networks," in *Eleventh International Conference on Mobile Data Management, MDM 2010, Kanas City, Missouri, USA, 23-26 May 2010*, T. Hara, C. S. Jensen, V. Kumar, S. Madria, and D. Zeinalipour-Yazti, Eds. IEEE Computer Society, 2010, pp. 75–84. [Online]. Available: \url{http://dx.doi.org/10.1109/MDM.2010.43}
48. L. Siksnys, J. R. Thomsen, S. Saltenis, M. L. Yiu, and O. Andersen, "A location privacy aware friend locator," in *Advances in Spatial and Temporal Databases, 11th International Symposium, SSTD 2009, Aalborg, Denmark, July 8-10, 2009, Proceedings*, ser. Lecture Notes in Computer Science, N. Mamoulis, T. Seidl, T. B. Pedersen, K. Torp, and I. Assent, Eds., vol. 5644. Springer, 2009, pp. 405–410. [Online]. Available: \url{http://dx.doi.org/10.1007/978-3-642-02982-0_29}
49. T. Sohn, A. Varshavsky, A. LaMarca, M. Y. Chen, T. Choudhury, I. E. Smith, S. Consolvo, J. Hightower, W. G. Griswold, and E. de Lara, "Mobility detection using everyday GSM traces," in *UbiComp 2006: Ubiquitous Computing, 8th International Conference, UbiComp 2006, Orange County, CA, USA, September 17-21, 2006*, ser. Lecture Notes in Computer Science, P. Dourish and A. Friday, Eds., vol. 4206. Springer, 2006, pp. 212–224. [Online]. Available: http://dx.doi.org/10.1007/11853565_13

50. M. Terrovitis, "Privacy preservation in the dissemination of location data," *SIGKDD Explorations*, vol. 13, no. 1, pp. 6–18, 2011.
51. The City of New York, "Taxi and Limousine Commission trip data," http://www.nyc.gov/html/tlc/html/about/trip_record_data.shtml, 2016.
52. The European Parliament and the Council of the European Union, "Article 25: Data protection by design and by default," http://ec.europa.eu/justice/data-protection/reform/files/regulation_oj_en.pdf, Apr. 2016.
53. "Uber technologies inc." https://www.uber.com/.
54. Volvo Car Group, "Volvo cars and uber join forces to develop autonomous driving cars," https://www.media.volvocars.com/global/en-gb/media/pressreleases/194795/volvo-cars-and-uber-join-forces-to-develop-autonomous-driving-cars, 2016.
55. Wikipedia, "Tesla model 3 — wikipedia, the free encyclopedia," https://en.wikipedia.org/w/index.php?title=Tesla_Model_3&oldid=750392389, 2016.
56. A. C. Yao, "How to generate and exchange secrets (extended abstract)," in *27th Annual Symposium on Foundations of Computer Science, Toronto, Canada, 27-29 October 1986*. IEEE Computer Society, 1986, pp. 162–167. [Online]. Available: \url{http://dx.doi.org/10.1109/SFCS.1986.25}
57. G. Zhong, I. Goldberg, and U. Hengartner, "Louis, lester and pierre: Three protocols for location privacy," in *Privacy Enhancing Technologies, 7th International Symposium, PET 2007 Ottawa, Canada, June 20-22, 2007, Revised Selected Papers*, ser. Lecture Notes in Computer Science, N. Borisov and P. Golle, Eds., vol. 4776. Springer, 2007, pp. 62–76. [Online]. Available: \url{http://dx.doi.org/10.1007/978-3-540-75551-7_5}

1 Interpolation optimizations

Before describing our optimizations, we recall the basics of polynomial interpolation.

For any given set of n coordinates, there exists a unique interpolating polynomial of degree $n+1$. When using a polynomial over a field, as in our context, an attacker gains no advantage from knowing $n-1$ points. *Lagrange interpolation* is used to evaluate a polynomial $p(x)$ of degree $n+1$ for any x, given n arbitrary points. For the input set (x_i, y_i), with $i \in \{0, 1, .., n\}$, we have:

$$p(x) = \sum_{j=0}^{n} \left(y_j \prod_{m=0, m \neq j}^{n} \frac{x_m - x}{x_m - x_j} \right)$$

From the above representation, it's easy to see that Lagrange interpolation requires $\mathcal{O}(n^2)$ time.

Decapsulation Here we exploit the following property of our specific application: trajectories are ordered sets of points, we do not need to try *every* set A' of size t, but only sets with consecutive points. In detail, if two trajectories (x_i^A, y_i^A) and (x_j^B, y_j^B) only have an intersection of size t if there exist a starting point s and a difference d such that $(x_{s+i}^A, y_{s+i}^A) = (x_{s+d+i}^B, y_{s+d+i}^A)$ for all $i = 0..t-1$.

For simplicity, consider the case where A contains $n = 2t$ points, and we need to find a sequence of t consecutive points that yields the correct interpolation. There's a total of $t+1$ sequences of the correct length, and A needs to attempt to verify each of them towards the check. Let each such sequence's dimension be denoted separately as

$$\overrightarrow{X^i} = \{x_i^A, x_{i+1}^A, .., x_{i+t}^A\} \cup \{x_0^h, x_1^h, .., x_t^h\}$$

$$\overrightarrow{Y^i} = \{y_i^A, y_{i+1}^A, .., y_{i+t}^A\} \cup \{y_0^h, y_1^h, .., y_t^h\}$$

For the first sequence, Decap computes a normal interpolation in $\mathcal{O}(n^2)$ time. For all following sequences, the interpolation at the x coordinates for X_k can be used to compute the interpolation of X_{k+1}. As shown in the following, this incremental step can be done in $\mathcal{O}(n)$ time. Note that the receiver is only interested in learning $p(0)$. The receiver saves the numerators and denominators from first run in two vectors $\overrightarrow{O^0}$ (over), $\overrightarrow{U^0}$ (under), given as:

$$\overrightarrow{O^0} = \left[\prod_{m=0, m\neq j}^{t} X_m^0 \,\middle|\, j \in \{0..t\} \right] \tag{4}$$

$$\overrightarrow{U^0} = \left[\prod_{m=0, m\neq j}^{t} \left(X_m^0 - X_j^0\right) \,\middle|\, j \in \{0..t\} \right] \tag{5}$$

Since each O_i^0 and U_i^0 take $\mathcal{O}(n)$ time to compute, Equation 4 and Equation 5 are both clearly $\mathcal{O}(n^2)$. The following describes how to compute O_i^{k+1} and U_i^{k+1} from O_i^k and U_i^k in linear time, for any $k \geq 0$. The transformation from $\overrightarrow{O^k}$ to $\overrightarrow{O^{k+1}}$ consists of two steps, the first for $j \in \{0..t-2\}$ and the second for $j \in \{t, .., n-1\}$. For the case when O_{t-1}^{k+1}, we note that is exactly the value O_0^k. For $j \in \{0..t-2\}$, the numerator is calculated as:

$$O_j^{k+1} = O_{j+1}^k \cdot \frac{X_t^{k+1}}{X_0^k}$$

Which intuitively means just including one new factor $X^{k+1}t$ and removing the factor X_0^k. E.g., for $k = 0$, this means removing the first x-coordinate x_0, and including x_{t+1}. For $j \in \{t..n-1\}$, a similar computation is carried out:

$$O_j^{k+1} = O_j^k \cdot \frac{X_t^{k+1}}{X_0^k}$$

Clearly, this is constant time for each of the n indexes, which means that the time for updating from $\overrightarrow{o}^k$ to $\overrightarrow{o}^{k+1}$ is $\mathcal{O}(n)$. For the denominators, the computations are more expensive, however the asymptotic runtime is the same. Similarly as for numerators, we have that for $j \in \{0..t-2\}$:

$$U_j^{k+1} = U_{j+1}^k \cdot \frac{X_t^{k+1} - x_j^{k+1}}{x_0^k - x_j^{k+1}}$$

For $j \in \{t..n-1\}$:

$$U_j^{k+1} = U_j^k \cdot \frac{X_t^{k+1} - X_j^{k+1}}{X_0^k - X_j^{k+1}}$$

As with the numerators, this is constant time per j, and thus runs in $\mathcal{O}(n)$. Now for the value of U_{t-1}^{k+1}, we have to recompute the entire denominator.

$$U_{t-1}^{k+1} = \prod_{m=0, m\neq j}^{t} \left(X_m^{k+1} - X_j^{k+1}\right) \Big| \, j \in \{0..t\}$$

This means that the update for the denominators take $\mathcal{O}(n+n) = \mathcal{O}(n)$. Further, for each $k \in \{0..t\}$, after $\overrightarrow{O^k}$ and $\overrightarrow{U^k}$ have been computed, the evaluation of the polynomial is needed. This is done as:

$$p(0) = \sum_{j=0}^{t} \left(y_j \frac{o_j}{u_j} \right)$$

Which runs in $\mathcal{O}(n)$ time. Thus, for $k = 0$ interpolation requires $\mathcal{O}(n^2 + n) = \mathcal{O}(n^2)$ work, which is followed by t evaluations that run in $\mathcal{O}(n)$ time. In total, the receiver thus spend $\mathcal{O}(n^2 + n^2) = \mathcal{O}(n^2)$ time.

Encapsulation The encapsulation algorithm Encap can not utilize incremental computation during the interpolation process. However, the denominator is in this case fixed, and only needs to be computed once.

$$\overrightarrow{u} = \left[\prod_{m=0, m\neq j}^{t} \left(x_m^B - x_j^B\right) \middle| j \in \{0..t\} \right]$$

The numerator needs complete recomputation for every hint:

$$\overrightarrow{O^x} = \left[\prod_{m=0, m\neq j}^{t} \left(x_m^B - x\right) \middle| j \in \{0..t\} \right]$$

It's possible to achieve a speedup for the numerator by precomputing

$$\overrightarrow{P} = \left[\frac{y_j}{u_j} \middle| j \in \{0..t\} \right]$$

Then, to compute the y-component of every hint, the sender computes

$$p(x_i^h) = y_i^h = \sum_{j=0}^{t} \left(P_j O_j^{x_i^h} \right)$$

However, the improvement of precomputing P does not improve the running time asymptotically.