MaxPace: Speed-Constrained Location Queries (Full Version)

Per Hallgren
Chalmers University of Technology, Sweden

Martín Ochoa
Singapore University of Technology and Design, Singapore

Andrei Sabelfeld
Chalmers University of Technology, Sweden

***Abstract*—With the increasing proliferation of mobile devices, location-based services enjoy increasing popularity. At the same time, this raises concerns regarding location privacy, as seen in many publicized cases when user location is illegitimately tracked both by malicious users and by invasive service providers. This paper is focused on privacy for the location proximity problem, with the goal of revealing the proximity of a user without disclosing any other data about the user's location. A key challenge is attacks by multiple requests, when a malicious user requests proximity to a victim from multiple locations in order to position the user by trilateration. To mitigate these concerns we develop MaxPace, a general policy framework to restrict proximity queries based on the speed of the requester. MaxPace boosts the privacy guarantees, which is demonstrated by comparative bounds on how the knowledge about the users' location changes over time. MaxPace applies to both a centralized setting, where the server can enforce the policy on the actual locations, and a decentralized setting, dispensing with the need to reveal user locations to the service provider. The former has already found a way into practical location-based services. For the latter, we develop a secure multi-party computation protocol that incorporates the speed constraints in its design. We formally establish the protocol's privacy guarantees and benchmark our prototype implementation to demonstrate the protocol's practical feasibility.**

I. Introduction

The increasing proliferation of mobile devices drives tremendous developments in the area of mobile computing. Mobile Internet usage already dominates over desktop both by the number of users [14] and time spent [6]. As part of these developments, location-based services enjoy increasing popularity, enabling location-based features such as finding nearby points of interest or discovering friends in proximity.

At the same time, services that involve location information raise increasing privacy concerns. These concerns apply to both protecting the privacy with respect to other users and with respect to service providers. There are publicized cases of both scenarios in practice. For the former scenario, the smartphone app *"Girls around me"* allowed users to find other users (profiled as female) who recently had checked in on Foursquare [3]. Deemed as a serious privacy violation, the app has since been banned from the Foursquare API and removed from the app store. For the latter scenario, the smartphone 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 [2].

These privacy concerns call for developing privacy-aware location based services [13], [28]. Accordingly, our goal is striving for rigorous guarantees for the protocols that underly practical location-based services.

The following motivates our approach. We start with unconstrained services that freely share user location and gradually illustrate the protection measures that need to be in place to protect against location privacy attacks. In the following paragraphs, let us focus on a scenario where a malicious user attempts to leverage a location-based service to attack location privacy of another user by looking at four techniques that have been applied in practice. Subsequently, we discuss the service-attacks-user scenario in a decentralized setting.

a) From positions to distances: Directly revealing locations may violate privacy. For example, as of May 2015, Facebook Messenger defaulted to sending user location tags with all messages, which was exploited by a "stalking" Chrome extension [12]. Since then, Facebook has deactivated location sharing from the desktop web page.

b) From distances to approximate distances: To aid privacy, the next step is to reveal distances instead of locations. For example, the dating app Tinder revealed distances to other users. It is straightforward to bypass this protection and calculate the location. Indeed, an illustrative attack on Tinder has been detailed by Include Security [29], revealing exact position of any user. At the core of this and other practical attacks is the technique of *trilateration*.

Trilateration uses multiple requests, where each results in learning that the user is located on a circle that is centered in the requester's position. Trilateration derives the user location as the intersection point of three circles, precisely pinpointing the user, as illustrated in Figure 1.

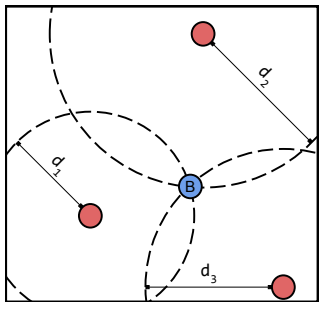

Fig. 1: Trilateration attack

c) From approximate distances to proximity: To mitigate trilateration attacks, some services have introduced ap-

proximate distance. For example, Facebook's Nearby Friends rounds the distance information. However, this mitigation can be easily bypassed. Recent research systematizes these attacks and identifies a number of location-based services where it is possible to reveal the user location even if distances are approximated/obfuscated [23], [15], [22].

d) From proximity to speed-constrained proximity: Next notch up for privacy is not to reveal the distance but to reveal proximity to the other user. This drastically reduces information about the location: it is only one bit per request. Still, proximity can be viewed as coarse-grained approximation, and attacks to pinpoint user location are still possible.

Instead of trilateration as in the case of distance-based attacks, the attacker in the proximity-based setting essentially (i) solves the *DiskCoverage* problem [22] by covering the plane with non-overlapping circles until getting a positive proximity response, and then (ii) solves the *DiskSearch* problem [22] to get the exact location by aiming to divide the constraining discs by half with each request.

In this paper, we explore in depth the effect of constraining the speed of the requester on the effect of the user discovery attacks. The key idea is that it is unrealistic for honest users to drastically change their location between subsequent requests. Hence, we aim at a policy that impedes the attacker without hampering practical usage of proximity protocols.

The *DiskCoverage* problem is in the focus of this work, as speed-constraining techniques as applied in practice provide little protection against an attacker on the *DiskSearch* problem.

e) MaxPace: speed-constrained proximity: We develop MaxPace, a general policy framework to restrict proximity queries based on the speed of the requester. The effect of speed constraints is illustrated in Figure 2. Each query corresponds to a disk. Large disk overlaps with speed-constrained queries means that the attacker learns little information from each query compared to the unrestricted attacker, and thus needs to issue more requests to learn the victims location.

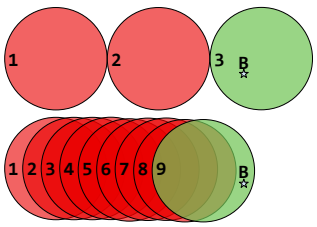

Fig. 2: Different protocols

MaxPace applies to both a centralized setting, where the server can enforce the policy on the actual locations, and a decentralized setting, dispensing with the trust to the service provider. In the centralized setting, we are encouraged by the recent changes in the policies of the popular centralized location-based services Facebook, Swarm, and Tinder [22] to incorporate forms of speed-based constraints. Our study is intended to provide rigorous analysis and understanding of guarantees provided by this type of constraints.

In the decentralized setting, we develop a secure multi-party computation protocol. This offers a contribution beyond the state of the art. The state-of-the-art protocols often lack formal privacy guarantees [32], [26], [30], [8]. Further, when there are formal guarantees a dominant assumption in the most recent literature on securing location proximity [32], [26], [30], [8], [18], [20], [25], [11] is the assumption of a single run. In contrast, our approach does not impose such an assumption, allowing us to reason about multi-run attacks.

Though this paper tackles malicious attackers, colluding attackers have been out of reach for the state-of-the-art work both in the single-run [32], [26], [30], [8], [18], [20], [25], [11] and also in the multi-run [22] setting. This justifies restricting the scope to non-colluding attackers in this paper. Also note that an attacker may be prevented from using multiple devices and/or multiple accounts by using authentication on top of the proposed solution.

The paper offers the following contributions. Section II presents MaxPace, the speed-constrained proximity disclosure policy. Section III shows the bounds provided by MaxPace and compares them to the classical unrestricted attacker. Section IV shows how MaxPace can be enforced without a trusted third party, granting privacy of the locations of both involved parties. Our enforcement is based on a novel secure multi-party computation protocol. We formalize the privacy guarantees of the protocol in Section V. Section VI presents benchmarks of our implementation of the protocol, demonstrating its practical feasibility.

II. The MaxPace Proximity Disclosure Policy

We introduce MaxPace, a policy for location proximity disclosure which limits the speed of a querying principal. The attacker is moving freely at arbitrary speed, but the victim is at a fixed position. The case when the victim is moving is an interesting field of study, but which poses challenges [4], left for future work.

The intuition is to force the attacker to behave as a benign user. In a normal setting, users assume any protocol participant behaves according to real-world physical constraints; other participants are e.g. walking, riding a bike, driving a car. The constraints imposed by MaxPace give less freedom to attackers during each query, causing them to learn less about the victim's location, and thus impose an additional effort to locate the victim. The exact benefits gained through MaxPace are discussed in detail in Section III, as compared to an unconstrained attacker on a proximity protocol.

When a principal wishes to know the proximity of another principal they issue a *location query*, as defined in Definition 1. The queried principal is henceforth referred to as Bob and the querying principal as Alice. That is, Alice asks Bob whether or not they are close. Alice may be malicious, and try to locate Bob by repeated querying.

Definition 1 (Location query)**.** *Let P be the set of possible coordinates (x, y) representing the position of a principal in the plane. A location query q is a tuple $(p, t) \in P \times \mathbb{N}$ where p is the position of the querying principal, and t the timestamp of when the query was issued.*

When *Bob* receives a query from *Alice*, he considers her speed. If *Alice* is respecting the maximum speed h set by the

policy, she receives the correct result. However, if she is moving too fast by quickly spoofing coordinates that are far apart she instead receives an unusable $\perp$-*value* as defined in Definition 2. Though MaxPace does not prevent usage artificial locations, it limits the effectiveness of such attacks.

Definition 2 ($\perp$-values)**.** *A value containing no useful information (such as an error message, a* `null`*-value or a freshly sampled uniformly random value) is called* $\perp$*.*

If a query respects the speed threshold, *Bob* computes the proximity, indicating *only* whether the principals are within a distance r. Distance between the points of two queries is calculated as described in Definition 3.

Definition 3 (Query distance)**.** *The Euclidean distance between two positions* p_1 *and* p_2 *is given by* $dist(p_1, p_2)$*.*

This paper uses a common definition of proximity [11], [22], defining proximity between two positions p_1 and p_2 as:

$$inProx(p_1, p_2, r) = \begin{cases} True & \text{if } dist(p_1, p_2) < r \\ False & \text{otherwise} \end{cases}$$

If *Alice* moves at a speed allowed by *Bob*, she may query for location proximity at any frequency. Once she surpasses this speed, any future requests she initiates yield $\perp$. This is further formalized in Definition 4.

Definition 4 (MaxPace)**.** *The responses* $L = \{l_1, l_2, ..., l_m\}$ *to a series of location queries* $Q = \{q_1, q_2, ..., q_m\}$ *from* Alice *to* Bob *respect MaxPace if and only if:*

$$l_i = \begin{cases} \perp & \text{if } \exists_{j<i} : \frac{dist(p_j, p_{j+1})}{time(q_j) - time(q_{j+1})} > h \\ inProx(p_i, p_B, r) & \text{otherwise} \end{cases}$$

where $q_i = (p_i, t_i)$ *and the position of* Bob *is* p_B*.*

Definition 5 (Query Time)**.** *Given a location query* $q = (p, t)$*, let* $time(q) = t$*.*

The two parameters r and h can be considered public and mutually agreed upon by *Alice* and *Bob* prior to running the protocol, e.g. as a part of the key exchange.

If a principal is detected as using a too high speed, they are seen as malicious and indefinitely prevented from learning further location information. However, as an effect of imprecise GPS positioning (e.g. by being under ground), or after having used a means of transportation not considered by the application (e.g. an airplane or high-speed train), the positions reported by a benign user may indicate that the user is traveling at higher speeds than allowed. A benign user can potentially be seen as malicious without having tried to attack the user. For applications in practice, some speed violations might need to be allowed in some cases. To reduce the period of time a benign principal is classified as malicious, a principal who previously has acted as if malicious can be forgiven and be allowed to query for location information again.

To forgive a principal, there are many viable strategies to reset the protocol. The simplest is arguably to reset after a fixed amount of time. An interesting approach could be to reset if the blocked principal returns to the point where it first broke the speed limit. This would capture the case when a user takes a flight abroad, and allow them to resume querying other principals after returning from the trip. If a user who has reported speeds in excess of those allowed is forgiven, the effort of a malicious attacker is lowered. In the worst case, the attacker knows exactly when a reset occurs and may then move at arbitrary speed between two queries. To what extent the attacker effort is affected is discussed in Section III-D.

III. Bounds on Attacker Effort

This section defines bounds for the attacker effort to locate the victim using a proximity disclosure protocol, both for the normal case and when applying the MaxPace policy. Disclosing only proximity forces the attacker to search a large portion of the plane to locate them. The bounds calculated here gives a measure of quickly the attacker can search the plane. The analysis considers the space as finite but of arbitrary size in discrete Euclidean coordinates, and the victim's position is a uniformly distributed variable. In this setting, for a fixed time period, the attacker's chance of finding the victim is negligible. Further, any holes left unexplored by the search strategy are of negligible size relative to the remaining area. Polakis et al. [22] present definitions for scenarios similar to the one considered in this paper, where an attacker is trying to locate a user in a finite section of the discrete Euclidean plane. Their terminology is reused in the following for clarity.

As mentioned previously, this work tackles an attacker who is trying to find which disk the user resides in, which is called the *DiskCoverage* problem. More precisely, *DiskCoverage* is defined in Definition 6. Note that the definition here is slightly different from the definition used by Polakis et al., even though the goal of the attacker is equivalent. In the original definition the attacker wants to minimize the time to completely cover a fixed space, while here the attacker attempts, but does not succeed, in solving the *DiskCoverage* problem in a fixed time and thus focuses on maximizing progress.

Definition 6 (*DiskCoverage*)**.** *The* DiskCoverage *problem is to find a set* S *containing the possible coordinates of the victim, such that* $|S| \leq r^2\pi$*.*

To calculate the effort for the attacker solve the *DiskCoverage* problem, the progress made with each individual query is needed. Herein, we say that the attacker *learns* an amount of knowledge from a location query, see Definition 7.

Definition 7 (Attacker knowledge per query)**.** *The neighborhood of radius* r *of a location query* $q = (p, t)$ *is a set* $cov_r(q)$*, s.t.* $\forall p_i \in cov_r(q) : inProx(p_i, p, r)$*. From the response of a single location query, the attacker learns if the victim's position* $p_b \in cov_r(q)$*. We thus call* $cov_r(q)$ *the attacker knowledge for query* q*.*

The *knowledge* of the attacker thus corresponds to the area (or set of points) which is within the proximity of any issued proximity request, and for a set Q of queries, the accumulated

knowledge is the union $\bigcup_{q \in Q} cov_r(q)$. For the attackers on both DecentMP and a plain protocol, an optimal attacker is assumed. For the plain protocol, the precise knowledge gained by the attacker can be calculated, while for DecentMP an upper bound on the knowledge is presented. Let the accumulated knowledge of an attacker that is not limited by MaxPace be called a_{plain} and the knowledge of an attacker limited by MaxPace be called $a_{MaxPace}$.

A. Knowledge of Unconstrained Attacker

Clearly any query by the unconstrained attacker which overlaps with previously covered areas (such that $cov_r(q_i) \cap cov_r(q_j) \neq \emptyset$) is a bad strategy, as it contains less knowledge (fewer points) than non-overlapping queries. The optimal attacker thus covers an area of $r^2\pi$ with each query. During T time units, the plain attacker does T/t_p queries at distinct locations, where t_p is the minimum time required to receive a response from a location query. This yields $a_{plain} = \pi r^2 \frac{T}{t_p}$.

B. Bounds for MaxPace

Now for an attacker constrained by MaxPace, for which an upper bound is given. Comparing the upper bound of an attacker on MaxPace to the attacker on the plain policy gives the *minimum* advantage MaxPace has over a plain policy.

Unlike the unrestricted querying policy, the optimal attacker on MaxPace is forced to query with overlapping coverages – otherwise they are travelling faster than the limit h and learn nothing at all. Note that the attacker may choose to query with a distance of $2 \cdot r$ with a large time interval to not have an overlap but the attacker gains more knowledge when querying as often as possible, as shown through Theorem 1.

Theorem 1 (Optimal attackers on MaxPace query as often as possible)**.** *Given two queries $q_s = (p_s, t_s)$ and $q_e = (p_e, t_e)$ which do not violate the MaxPace policy, and where $p_s \neq p_e$. If it is possible to define a third query $q_i = (p_i, t_i)$ such that for t_i and p_i $(t_s < t_i < t_e) \vee (p_s \neq p_i \neq p_e)$ holds, and q_s, q_i, q_e comply with MaxPace, then issuing the three queries q_s, q_i, q_e yields more information than issuing q_s, q_e.*

Proof. By contradiction, assume that $cov_r(q_e) \cup cov_r(q_s)$ is equal to $cov_r(q_e) \cup cov_r(q_i) \cup cov_r(q_s)$. This implies that either $cov_r(q_s) = cov_r(q_i)$ or $cov_r(q_e) = cov_r(q_i)$, which in turn implies $p_i = p_e \vee p_i = p_s$ ↯. □

From Theorem 1, the attacker sends a location query as soon as the policy allows them after moving one distance unit, thus waiting at most $s = 1/h$ time units between each query. The coverage for each query is calculated as the area of a circle of radius r, subtracting the area of the intersection with the previous query. How to calculate the area of circle intersections is given in [19]. The knowledge gained by an adversary for each query after the first one is given in Equation (1).

$$\pi r^2 - \left(2r^2 cos^{-1}\left(\frac{1}{2r}\right) - \frac{1}{2}\sqrt{4r^2 - 1}\right) \quad (1)$$

TABLE I: Speeds in m/s and km/h for the used scenarios

Activity	Walking	Running	Cycling	Bus	Car (highway)
m/s	2	3	5	14	33
km/h	7.2	10.8	18	50.4	118.8

TABLE II: Bounds for different speed radiuses

Speed	Radius			
	10	25	50	100
Walking	78.2	194.3	384.4	752.7
Running	52.2	130.0	258.1	508.8
Cycling	31.4	78.2	155.7	308.8
Bus	11.2	28.0	55.9	111.5
Car	4.8	11.9	23.8	47.5

For a more concise bound, simplifications are made to over-approximate Equation (1). This means an under-approximation of Equation (2) and over-approximation of Equation (3).

$$-2r^2 cos^{-1}\left(\frac{1}{2r}\right) \quad (2) \qquad \frac{1}{2}\sqrt{4r^2 - 1} \quad (3)$$

Note that $\lim_{x \to 0} cos^{-1}(x) = \frac{\pi}{2}$ (as $r \geq 1$). Thus, an under-approximation of Equation (2) is $-2r^2 \frac{\pi}{2}$. Equation (3) can be simplified by approximating $\sqrt{4r^2 - 1}$ to $\sqrt{4r^2}$. The concise over-approximation of Equation (1) is given in Equation (4).

$$\pi r^2 - \left(2r^2 \frac{\pi}{2} - \frac{1}{2} 2r\right) = \pi r^2 - 2r^2 \frac{\pi}{2} + \frac{1}{2} 2r = r \quad (4)$$

The attacker is able to perform a total of T/s queries. Including the first query, which covers an area of $r^2\pi$, the final area covered during *DiskCoverage* is given by:

$$a_{MaxPace} = r\left(\frac{T}{s} - 1\right) + r^2\pi$$

C. Comparisons

To evaluate the policy, the example activities of walking, running, cycling, riding a bus, and driving a car are considered, as listed in Table I. To compare a_{plain} and $a_{MaxPace}$, the ratio $\frac{a_{plain}}{a_{MaxPace}}$ is considered. Given the above example speeds and reasonable values of r, consider Table II. The table shows up to 753 times less information disclosure, demonstrating the effectiveness of MaxPace in practical scenarios. The value of t_p is chosen as 200 milliseconds for the plain protocol.

D. MaxPace with Resetting

As highlighted in Section II, there are scenarios where MaxPace is too restrictive. In these cases, it is beneficial if the protocol can be reset. When a reset occurs, an attacker will be able to reposition themselves independently of previous queries. If during a time frame T the attacker performs e resets, the bound of the attacker knowledge is $r\left(\frac{T}{s} - e\right) + \pi r^2 e$.

Concretely, consider a person who is walking and querying a radius of 100 meters, where the protocol is reset every 15 minutes, the time unit is seconds, and where $T = 3600$ (one hour). The attacker on the plain policy covers exactly 565486677.6 square meters. MaxPace without resetting restricts coverage to at most $751315.9m^2$, and MaxPace with resetting

gives a coverage of at most $876579.6m^2$. Though resetting in this case causes over 16% extra information leakage, even with resetting MaxPace yields with the given parameters 645 times less information than the plain protocol.

IV. Enforcement without Trusted Third Party

As foreshadowed earlier, MaxPace can be implemented in a straightforward way using a trusted third party who stores and manages location information for all users who are utilizing the service. Any already existing service can easily deploy MaxPace as an additional privacy measure. Many applications scenarios lack a natural third party that can be trusted, and a decentralized trust-model has obvious benefits as compared to giving location information to third parties. Services are usually not deployed in a decentralized manner without trusted parties, as for most application scenarios there are no ad-hoc solutions readily available.

This section describes how MaxPace can be enforced using a *Secure Multi-party Computations* (SMC) protocol without a trusted third party. The concrete protocol is referred to as DecentMP (short for *Decentralized MaxPace*).

A. Secure Arithmetics for SMC

There are a variety of primitives for implementing SMC, including garbled circuits [31], partial [21] and fully homomorphic encryption schemes [9] among others. To instantiate the MaxPace policy without third parties, we chose the BetterTimes system by Hallgren et al. [10]. This construction gives privacy guarantees against a malicious *Alice* Further, being based on additively homomorphic cryptography, it supports storing intermediate values from previous computations, a central feature in the implementation of MaxPace. Additionally, there is an open and efficient implementation of this construction that allows us to benchmark our results and discuss the applicability of our protocol in practice. For the scope of this paper, Alice holds the private key for all BetterTimes computations and is the only principal able to decrypt data. However, *Bob* is able to perform arithmetic computations using the BetterTimes system.

A BetterTimes formula is composed of arithmetic instructions. BetterTimes-instructions are recursive data structures, An instruction is either a binary operation (e.g. addition or subtraction), for which each operand is another instruction, or a scalar value. The formula is evaluated by *Bob*, and all actions taken by *Alice* are implicitly determined by any messages *Bob* sends. If *Alice* deviates from the protocol while computing a BetterTimes formula the output is a uniformly random number, and *Alice* learns only $\perp$. The guarantees of the construction are discussed further in Section V.

1) EasyTimes, more readable BetterTimes-syntax: This section details a subset of python, called *EasyTimes*, which directly maps into the syntax of BetterTimes. EasyTimes allows arithmetic formulas to be expressed in a more readable and concise manner than the original syntax. The full translation is detailed in Appendix D. In short, BetterTimes-instructions are simply represented by normal addition, subtraction and multiplication operations. The operations are overloaded, and when used a formula is constructed in the background, which later can be evaluated. Outputs are in the new syntax marked using calls to the `output()` function.

All coin tosses can be sampled from a cryptographically secure source using the $random(start, end)$ function. By convention, variables storing a ciphertext uses a prefixing "$c\_$". As in normal Python, exponentiation is written using double multiplication signs; x^y is written as `x ** y`.

2) Extension to BetterTimes for multiple outputs: As an additional contribution of this work, an extension to BetterTimes is constructed which allows for a single formula to yield multiple outputs. The construction is presented in detail in Appendix A. In short, the extension provides means for utilizing more than one `output()` call.

B. Homomorphic primitives

Below, the building blocks needed to construct DecentMP are described before proceeding to present the full protocol.

1) Homomorphic Distance: The squared Euclidean distance (henceforth simply called distance) can be computed using additively homomorphic encryption in a privacy-preserving manner [32], [7], [25], [24], [11]. As shown in [10], most approaches are only secure in the semi-honest model but can be made secure in the malicious model using BetterTimes.

The approaches above require that *Bob* holds one of the two coordinates in the clear. Listing 1 shows a short protocol in EasyTimes syntax which computes the distance between (x_1, y_1) and (x_2, y_2) without any plaintext knowledge. Computing the distance while holding a coordinate in the plain is similar, however where the last two parameters `c_x2, c_y2` are plaintexts and thus have a different type. For the scope of this paper such a method is called $ODist_{plain}$.

Listing 1: Procedure for distance computation

```
def ODist(c_x1, c_y1, c_x2, c_y2):
    c_sq1 = c_x1 * c_x1 + c_y1 * c_y1
    c_sq2 = c_x2 * c_x2 + c_y2 * c_y2
    c_cross = c_x1 * c_x2 + c_y1 * c_y2
    return c_sq1 + c_sq2 - 2 * c_cross
```

2) Homomorphic Comparisons: There are several solutions to compute comparisons homomorphically in the literature [11], [7] by making use of bit parity. Here, a comparison method very similar to the one by Hallgren et al. is used [11].

Hallgren et al. use the fact that $(x - y) \cdot \rho$, with ρ uniformly random, yields $\perp$ if and only if x and y are not equal. Thus, to compare $x < y$, it's possible to check if $\exists i \in \{0..y-1\} : (x - i) \cdot \rho = 0$. However, where Hallgren et al. use an array of equality-checks and shuffle it to hide which slot is equal to the compared value, instead the values are multiplied here, as shown in Listing 2.

Listing 2: Procedure for computing "less than"

```
def lessThan(c_x, y):
    c_l = 1
    for i in range(0, y - 1):
        c_l = c_l * (c_x - i)
```

```
    return c_l * random(1, k)
```

3) Homomorphic Proximity Check: To enforce the MaxPace policy, it is necessary to compute whether two points are near each other. In short, the formula consists of chaining $ODist_{plain}$ and `lessThan`, as shown in Listing 3.

Listing 3: Procedure to check the proximity of two points

```
def proximity(c_x1, c_y1, x2, y2, r):
    dist = ODist_plain(c_x1, c_y1, x2, y2)
    return lessThan(dist, r ** 2)
```

4) Homomorphic Speed: The following shows, to the best of the authors' knowledge, the first case where speed computations are used together with additively homomorphic encryption. More precisely, *Bob* calculates whether or not the speed of *Alice* is under an allowed threshold, as shown in Listing 4.

Listing 4: Procedure to check for too fast movement

```
def speed(c_x1, c_y1, c_x2, c_y2, h, t):
    dist = ODist(c_x1, c_y1, c_x2, c_y2)
    return lessThan(dist, (h * t) ** 2)
```

The speed when moving d distance over a time t is computed as d/t. In MaxPace, the goal is to check when the speed exceeds a threshold h. Thus, the sought computation is $\frac{d}{t} \leq h$, which can be re-written (for non-negative integers) as $d \leq h \cdot t$.

C. DecentMP

The protocol is shown in Listing 5. The procedure is executed by *Bob*, while operations carried out by *Alice* are implicitly determined through the BetterTimes system.

Listing 5: Request handling using DecentMP

```
def mpRequest(ev,c_xA,c_yA,xB,yB,h,r,cache):
  formula = SecureFormula(ev, h,r, c_xA, c_yA,
              xB, yB, cache['a'], cache['t'],
              cache['x'], cache['y'])
  with formula as sf:
    c_xA,c_yA,xB,yB,h,r,ct,c_ca,c_cx,c_cy
          = sf.inputs
    t = now()
    pr = proximity(c_xA, c_yA, xB, yB, r)
    if 'x' in cache:
      v = speed(c_xA,c_yA,c_cx,c_cy,h,t-ct)
      alpha = random(1, k) * (v + c_ca)
      sf.output(pr + alpha)
      sf.output(alpha)
    else:
      sf.output(pr)

  out = formula.evaluate()
  c_result = out[0]

  cache['a'] = out[1] if 'x' in cache else 0
  cache['t'] = t
  cache['x'] = c_xA
  cache['y'] = c_yA
  return c_result
```

For the first run, the protocol simply returns the proximity result and caches the query's position and time. *Bob* also initializes a special cache value a which is used to accumulate all speed threshold checks. For following requests, the speed threshold v is combined with the accumulated speed threshold. By adding the proximity result to the accumulated speed threshold, *Bob* constructs `c_result`. Note that all values depending on *Alice*'s inputs are encrypted and not readable by *Bob*.

V. Privacy Guarantees of DecentMP

This section shows that DecentMP provides very strong privacy guarantees. The central privacy notion of this work is according to Definition 8, following the standard SMC security definitions of [16] against malicious adversaries.

Definition 8 (Privacy definition)**.** *A protocol π is said to privately implement a functionality g against malicious adversaries if for every adversary $\mathcal{A}$ against the protocol π, there exist a simulator $\mathcal{S}$ such that:*

$$\{\text{IDEAL}_{g,\mathcal{S}}(\overrightarrow{x}, \overrightarrow{y})\} \stackrel{c}{\equiv} \{\text{REAL}_{\pi,\mathcal{A}}(\overrightarrow{x}, \overrightarrow{y})\}$$

where $\stackrel{c}{\equiv}$ denotes computational indistinguishability of distributions.

For space reasons, we recall the IDEAL and REAL constructions and the computational indistinguishability definitions in Appendix B. The intuition behind this definition is that the implementation of g by π should be as secure as an ideal implementation of g using a third party. The desired functionality for DecentMP is specified by Definition 9.

Definition 9 (Constrained speed querying functionality)**.** *The functionality of a speed-constraining functionality g is a function from queries to responses: $g : Q \to L$.*

$$g(q_1, \ldots, q_m)[i] = \begin{cases} \perp & \text{if } \exists_{j<i} : \frac{dist(p_j, p_{j+1})}{time(q_j) - time(q_{j+1})} > h \\ inProx(p_i, p_b, r) & \text{otherwise} \end{cases}$$

where $q_i = (p_i, t_i)$ and p_b is the position of Bob.

Bearing the functionality Definition 9 in mind, recall the protocol resulting from the formula shown in Listing 5. Combining the two, the privacy-guarantees sought for DecentMP are captured by Theorem 2.

Theorem 2 (Privacy guarantees of DecentMP)**.** *The protocol π resulting from evaluating the program Listing 5 implements the functionality of Definition 9 privately according to Definition 8.*

A. Proofs

Now, to prove that DecentMP is secure according to Definition 8, we need to show: **a)** the protocol implements the desired functionality, that is, when used by honest parties, it implements the functionality g of Definition 9; and **b)** if misused by a malicious *Alice*, we can simulate *Alice*'s view based exclusively on her inputs and the outputs of the g.

a) Proof of correct functionality: First, the functionality must be privacy-preserving in the presence of only benign parties. This is captured by Theorem 3, for which the following proof shows that DecentMP is indeed privacy-preserving in the absence of malicious adversaries.

Theorem 3. *DecentMP implements the functionality g as defined in Definition 9*

Proof. By construction, if `alpha` (henceforth α) evaluates to the encryption of 0, then the result of the proximity request is correctly computed and disclosed to *Alice*, building on the correctness of the proximity computation protocol. `cache['a']` initially encrypts 0, and remains constant if and only if every invocation of `speed` returns the encryption of 0, which is the case when the speed limitation is respected. The first time that the speed limitation is violated, α is not an encryption of 0, and thus `c_result` encrypts $\perp$. `cache['a']` accumulates the sum of the previous evaluations of α. Since α evaluates to the encryption of either $\perp$ or 0, after the first speed violation `cache['a']` is an encryption of $\perp$. □

b) Proof of secure joint computation: Now for the more interesting case, when adversaries deviate from the protocol to try to infer additional data about the victim. This is captured by Theorem 2, for which the proof is presented in the following. The intuition behind the proof of Theorem 2 is as follows. DecentMP defines α as an arithmetic formula. From the security guarantees of [10], it follows that a malicious *Alice* that tampers with the protocol at any point up to the evaluation of α, will cause α to encrypt $\perp$. This in turn will cause the proximity result sent to *Alice* to be random, and cause `cache['a']` to be updated just as in the case where *Alice* does not respect the MaxPace speed policy, making subsequent location responses yield an encryption of $\perp$.

Since we leverage on primitives of BetterTimes to build DecentMP, we recall the privacy guarantees we obtain from using this construction. The privacy guarantees of [10] against malicious adversaries can be summarized as follows. In this setting, the possessor of the private key (*Alice*) is considered potentially malicious, whereas the party performing homomorphic operations on encrypted data (*Bob*) is considered to be honest. Indeed, as it is usual the case with protocols based on partially homomorphic cryptography, in this setting *Alice* gets the result of the joint computation, which by construction *Bob* cannot learn. *Bob* could still sabotage the outcome of the computation, but no such adversary is considered in this setting, since our focus is on *privacy* guarantees. Now, if *Alice* would tamper with the protocol to try to learn more about the private inputs of *Bob* than allowed by the arithmetic formula (the functionality), BetterTimes guarantees that she instead receives a fresh uniformly random value.

The main theorem of BetterTimes is proved by showing that all partial computations outsourced to *Alice* are independent and uniformly random, and the final value of the formula is $\perp$ if *Alice* does not comply with the protocol, and the correct output of the formula otherwise. This is captured by Lemma 1. The proof of Lemma 1 is presented in Appendix C.

Lemma 1 (Fundamental lemma of BetterTimes)**.** *For a fixed but arbitrary arithmetic formula $g(\overrightarrow{x}, \overrightarrow{y})$ represented by a recursive instruction $\iota \in$ **Ins** against the protocol π resulting from $evaluate(\iota)$, all intermediate messages to* Alice *are independent and uniformly random, and the last message $result$ of the protocol is an encryption of the output of $g(\overrightarrow{x}, \overrightarrow{y})$ if* Alice *is honest, and an encryption of a uniformly random value otherwise.*

However, Lemma 1 only shows that the final result is secured. Now, to show that also for intermediate values added to the output are secure, Lemma 1 can be extended to Lemma 2. The proof of Lemma 2 is found in Appendix C.

Lemma 2 (Extension of Fundamental lemma of BetterTimes)**.** *For a fixed but arbitrary arithmetic formula $g(\overrightarrow{x}, \overrightarrow{y})$ represented by a recursive instruction ι constructed using EasyTimes against the protocol π resulting from $evaluate(\iota)$, all intermediate messages to* Alice *are independent and uniformly random, and the final result and intermediate output values of the protocol are an encryption of $\perp$ for a dishonest* Alice*.*

Proof of Theorem 2. First, note that from the extension to the BetterTimes system as detailed in Appendix A and from Lemma 2, Corollary 1 follows immediately.

Corollary 1. *For the arithmetic formula represented by the recursive instruction ι resulting from Listing 5, all intermediate values sent to* Alice *are encryptions of $\perp$. The intermediate output value from the evaluation of α and the final result are encryptions of $\perp$ if* Alice *deviates from the protocol.*

After performing m location queries *Alice* has observed the intermediate values in each query q_i and the respective location response l_i by *Bob*. From Corollary 1, and by construction of the `cache['a']`, it follows that if *Alice* cheats for the first time when jointly computing l_i with *Bob*, then $l_i = \perp$ and $\forall_{j>i}\ l_j = \perp$. Further, from Corollary 1, it follows directly that all intermediate values in a joint computation, denoted by $\overrightarrow{v}_i$, are equal to $\perp$. Without loss of generality, let's assume that a class of malicious adversaries $\mathcal{A}_x$ are dishonest when jointly computing the location response l_x. The output of any such malicious $\mathcal{A}_x$ against DecentMP can be simulated by a simulator $\mathcal{S}_x$ that outputs:

$$\mathcal{S}_x(q_1, \ldots, q_m, l_1, \cdots l_m) = \mathcal{A}_x(q_1, \ldots, q_m, l'_1, \cdots l'_m, \overrightarrow{v}_1, \cdots \overrightarrow{v}_m)$$

The outputs l'_i corresponding to the view of $\mathcal{A}$ in a real execution are easy to simulate, as they can be computed using only the inputs through:

$$l'_i = \begin{cases} \perp & \text{if } i \geq x \\ l_i & \text{otherwise} \end{cases}$$

And the intermediate messages can be simulated as $\overrightarrow{v}_j = \perp, \cdots, \perp$ as per Lemma 2.

□

VI. Implementation and Benchmarks

This section describes the implementation of DecentMP and presents benchmarks from initial experiments. The prototype was done in Python using the library provided by [10]. The results show that it is feasible to use MaxPace in a decentralized setting for practical scenarios. Five typical scenarios were used, to give data on different speed thresholds (h). Four different values of the proximity threshold are measured (r).

The data shows that the protocol scales reasonably well in both h and r. Arguably, most configurations may be applicable to real applications already in its current state. Several configurations using the prototypical implementation finish within 500 milliseconds, and many applications can load data in the background and do not require instant feedback and would thus be able to use the more expensive settings.

A. System Overview

The protocol is implemented straight forward from the description in Section IV, with only one optimization effort. Instead of multiplying all values in the `lessThan` function while computing the proximity, the original idea of [11] to encode the values as an array is used. This reduces the number of round-trips and is securely realized by caching subtrees in the formula (which is needed since the distance is reused).

The additively homomorphic cryptographic scheme used was the DGK scheme [5]. For the DGK scheme, the plaintext space is chosen separately from the key size. For decryption to be efficient a table of the size of the plaintext space is saved. A larger plaintext space means that more RAM is needed and also has some costs in terms of performance. The implementation keeps this table in memory, which gives a practical limitation due to RAM consumption (for 2048 bit keys, 22 bits of plaintext space requires 8192 MB RAM).

The plaintext space size is relevant in the context of location based services. As an over-approximation, consider a square with sides equal to the earth's circumference $4 \cdot 10^7$ meters. A coordinate is then $log_2(4 \cdot 10^7) \approx 25$ bits. Thus, 25 bits of plaintext space is needed to measure the earth with 1 meter resolution. 22 bits gives a precision of 5 meters for the earth.

The time between requests where not considered as attackers can be assumed to query often. A benign user may query rarely, but performance issues with large time spans for benign users can be resolved with a resetting strategy.

The benchmarks were performed on a single machine with 16MB RAM and an Intel i7-4790 CPU at 3.60GHz. Both the client and the server application were hosted locally, and were noted to be performing work of the same order of magnitude.

B. Performance

Benchmarks were carried out with a key of sizes 2048 bits, and plaintext space 22 bits. Figure 3 visualizes how the time of a single protocol execution time is affected by different configurations of h and r. Though configurations modelling higher speed and a larger radius cause longer protocol execution

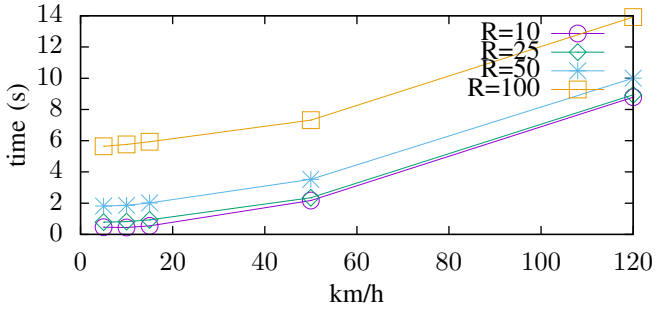

Fig. 3: Different speeds and proximity thresholds

times, in both of the parameters r and h, the time to complete the protocol grows less than quadratic.

Benchmarks where also performed for a much smaller plaintext space of 10 bits. The results show similar performance, with a difference of at most 200 milliseconds for any scenarios, compared to 22 bits of plaintext space. Given a machine with more RAM, a higher precision can be utilized without noticeably affecting user experience.

C. Communication Cost

The protocol may incur a significant communication cost. The number of round trips is dictated by the speed threshold, and the size of the messages depends on the key size. Further, the size of the result array is affected by the proximity threshold. Table III shows how different values of r and h affect communication. These numbers exclude the additional overhead of the structure of the messages, which is not significant.

TABLE III: Communication cost in kilobytes (messages)

Activity	Proximity Threshold (meters)				Messages
	10	15	50	100	
Walking	166	610	2050	7392	11
Running	196	640	2080	7422	17
Cycling	286	730	2170	7512	35
Bus	1076	1520	2960	8302	193
Car	4706	5150	6590	11932	919

Communication cost ranges from 166 kilobytes to 12 megabytes. 12 MB is rather a lot of data for geometric computations, but seeing as most devices can handle high-quality video streaming, all results are within practical applicability.

VII. Related Work

Location privacy is a well recognized issue, as seen from a diverse range of surveys: [13], [28], [17]. Protecting location disclosure during continuous queries through speed limitation has been applied in practice [22], but to the best of the authors' knowledge this is the first formalization of such approaches and the first which quantifies attacker effort in these cases. There are several active research areas which touch upon different components of this work. The following positions this work in relation to the more relevant neighbouring approaches.

Within location privacy, different works protect different parts of a user's data. Many approaches provide k-anonymity [27], [17], where the location of the user is indistinguishable among a set of users, where the primary objective is

to protect the identity from the attacker. This work protects the location, and does not consider privacy of the identity. Works of this type, though similar, are orthogonal to MaxPace.

Considering moving principals is a key feature of this work. To the best of the authors' knowledge, there is no literature on how to maintain privacy if both the attacker and the victim are moving. Within location proximity specifically, there is a fair amount of work [32], [25], [8], [18], [26], [20]. However, the majority of current research focuses on static principals. A practical application requires security over continuous queries.

One countermeasure for when the attacker is moving is mentioned by Narayanan et al. as an effect of their construction [20]. By mapping each principal to a grid cell, called a *cloaking region*, and calculating distances between the cloaking regions, nothing more than the region can be leaked. That is, it's impossible to solve the *DiskSearch* problem.

There are several drawbacks with cloaking regions. Foremost, a significant chance of both false positives and negatives, as illustrated in Figure 4. Further, as highlighted by Cuellar et al. [4], if an attacker samples the victims location as the victim changes region, they know that the victim is close to the region's border. In general, simply making use of cloaking regions have no effect on the *DiskCoverage* problem, as is the focus of MaxPace.

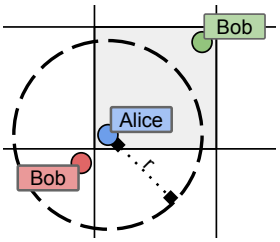

Fig. 4: Precision issues

Polakis et al. [22] investigate different disclosure strategies employed in the wild, such as disclosing distances or rounded distances. For several popular social networks, they perform measurements of how quickly different attack strategies can solve both the *DiskCoverage* and *DiskSearch* problems. Polakis et al. advocate using a cloaking region to improve privacy. As mentioned earlier, this helps to some extent when the victim is static for the *DiskSearch* problem, but has no effect on the *DiskCoverage* problem.

Lastly, one prime feature of MaxPace is that it is possible to deploy without a trusted third party. As highlighted above, trust may be used to enforce any policy with low computational effort, but such approaches have trouble when the aim is formal privacy guarantees [17].

VIII. Conclusions

We have developed MaxPace, a framework for speed-constrained location queries. We have demonstrated the advantages over unrestricted location queries by comparing bounds on an attacker's knowledge. The framework is susceptible to both centralized and decentralized deployment. The former has already found a way into practical location-based services. For the latter, we have devised a speed-constrained secure multi-party computation protocol for location proximity and formally established its privacy guarantees. We have reported on experiments with a prototype implementation which shows that the protocol can be used in practice.

Our knowledge bounds focus on the disk coverage problem as the main contributor to an attacker's knowledge. A study of the disk search problem is subject to future work. In addition, we are interested in further developing the resetting strategies outlined in Section II. This will allow MaxPace to deal with the imprecision of GPS and high-speed transportation. Finally, we plan to investigate scenarios where, in addition to Alice, Bob may also move in-between requests. While Bob's movement makes trilateration more difficult, more work is needed to quantify how Bob's movements affect the privacy guarantees.

Acknowledgments This work was funded by the European Community under the ProSecuToR project and the Swedish research agencies SSF and VR.

References

[1] G. Barthe, B. Grégoire, and S. Zanella Béguelin. Formal certification of code-based cryptographic proofs. POPL '09, pages 90–101, New York, NY, USA, 2009. ACM.

[2] 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.

[3] D. Coldewey. "Girls Around Me" Creeper App Just Might Get People To Pay Attention To Privacy Settings. http://techcrunch.com/2012/03/30/girls-around-me-creeper-app-just-might-get-people-to-pay-attention-to-privacy-settings/, Mar. 2012.

[4] J. Cuéllar, M. Ochoa, and R. Rios. Indistinguishable regions in geographic privacy. In *SAC 2012, Riva, Trento, Italy, March 26-30, 2012*, pages 1463–1469, 2012.

[5] I. Damgård, M. Geisler, and M. Krøigaard. Efficient and secure comparison for on-line auctions. In *ACISP 2007, Townsville, Australia, July 2-4, 2007, Proceedings*, pages 416–430, 2007.

[6] K. Dreyer. Mobile Internet Usage Skyrockets in Past 4 Years to Overtake Desktop as Most Used Digital Platform. http://www.comscore.com/Insights/Blog/Mobile-Internet-Usage-Skyrockets-in-Past-4-Years-to-Overtake-Desktop-as-Most-Used-Digital-Platform, Apr. 2015.

[7] Z. Erkin, M. Franz, J. Guajardo, S. Katzenbeisser, I. Lagendijk, and T. Toft. Privacy-preserving face recognition. In *PETS 2009, Seattle, WA, USA, August 5-7, 2009. Proceedings*, pages 235–253, 2009.

[8] D. Freni, C. R. Vicente, S. Mascetti, C. Bettini, and C. S. Jensen. Preserving location and absence privacy in geo-social networks. In *CIKM 2010, Toronto, Ontario, Canada, October 26-30, 2010*, pages 309–318, 2010.

[9] C. Gentry. Fully homomorphic encryption using ideal lattices. In *STOC*, pages 169–178, 2009.

[10] P. A. Hallgren, M. Ochoa, and A. Sabelfeld. Bettertimes - privacy-assured outsourced multiplications for additively homomorphic encryption on finite fields. In *ProvSec 2015, Kanazawa, Japan, November 24-26, 2015, Proceedings*, pages 291–309, 2015.

[11] P. A. Hallgren, M. Ochoa, and A. Sabelfeld. Innercircle: A parallelizable decentralized privacy-preserving location proximity protocol. In *PST 2015, Izmir, Turkey, July 21-23, 2015*, pages 1–6, 2015.

[12] A. Khanna. Stalking Your Friends with Facebook Messenger. https://medium.com/faith-and-future/stalking-your-friends-with-facebook-messenger-9da8820bd27d, May 2015.

[13] J. Krumm. A survey of computational location privacy. *Personal and Ubiquitous Computing*, 13(6):391–399, 2009.

[14] A. Lella. Number of Mobile-Only Internet Users Now Exceeds Desktop-Only in the U.S. https://www.comscore.com/Insights/Blog/Number-of-Mobile-Only-Internet-Users-Now-Exceeds-Desktop-Only-in-the-U.S, Apr. 2015.

[15] M. Li, H. Zhu, Z. Gao, S. Chen, L. Yu, S. Hu, and K. Ren. All your location are belong to us: breaking mobile social networks for automated user location tracking. In *MobiHoc'14, Philadelphia, PA, USA, August 11-14, 2014*, pages 43–52. ACM, 2014.

[16] Y. Lindell and B. Pinkas. Secure multiparty computation for privacy-preserving data mining. *IACR Cryptology ePrint Archive*, 2008:197, 2008.

[17] E. Magkos. Cryptographic approaches for privacy preservation in location-based services: A survey. *IJITSA*, 4(2):48–69, 2011.

[18] 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.*, 20(4):541–566, 2011.
[19] W. Mathworld. Circle-circle intersection, 2008.
[20] A. Narayanan, N. Thiagarajan, M. Lakhani, M. Hamburg, and D. Boneh. Location privacy via private proximity testing. In *NDSS 2011, San Diego, California, USA, 6th February - 9th February 2011*, 2011.
[21] P. Paillier. Public-key cryptosystems based on composite degree residuosity classes. In *Advances in Cryptology EUROCRYPT 99*, volume 1592, pages 223–238. Springer, 1999.
[22] I. Polakis, G. Argyros, T. Petsios, S. Sivakorn, and A. D. Keromytis. Where's wally?: Precise user discovery attacks in location proximity services. In *CCS, Denver, CO, USA, October 12-6, 2015*, pages 817–828, 2015.
[23] G. Qin, C. Patsakis, and M. Bouroche. Playing hide and seek with mobile dating applications. In *ICT Systems Security and Privacy Protection - IFIP SEC 2014, Marrakech, Morocco, June 2-4, 2014*, pages 185–196. Springer, 2014.
[24] A. Sadeghi, T. Schneider, and I. Wehrenberg. Efficient privacy-preserving face recognition. In *Information, Security and Cryptology - ICISC 2009, 12th International Conference, Seoul, Korea, December 2-4, 2009, Revised Selected Papers*, pages 229–244, 2009.
[25] J. Sedenka and P. Gasti. Privacy-preserving distance computation and proximity testing on earth, done right. In *ASIA CCS '14, Kyoto, Japan - June 03 - 06, 2014*, pages 99–110, 2014.
[26] L. Siksnys, J. R. Thomsen, S. Saltenis, M. L. Yiu, and O. Andersen. A location privacy aware friend locator. In *SSTD 2009, Aalborg, Denmark, July 8-10, 2009, Proceedings*, pages 405–410, 2009.
[27] N. Talukder and S. I. Ahamed. Preventing multi-query attack in location-based services. In *WISEC 2010, Hoboken, New Jersey, USA, March 22-24, 2010*, pages 25–36, 2010.
[28] M. Terrovitis. Privacy preservation in the dissemination of location data. *SIGKDD Explorations*, 13(1):6–18, 2011.
[29] M. Veytsman. How i was able to track the location of any tinder user. http://blog.includesecurity.com/2014/02/how-i-was-able-to-track-location-of-any.html, Feb. 2014.
[30] L. Šikšnys, J. R. Thomsen, S. Saltenis, and M. L. Yiu. Private and flexible proximity detection in mobile social networks. In *Mobile Data Management*, pages 75–84, 2010.
[31] A. C.-C. Yao. Protocols for secure computations. In *FOCS*, volume 82, pages 160–164, 1982.
[32] G. Zhong, I. Goldberg, and U. Hengartner. Louis, lester and pierre: Three protocols for location privacy. In *PET 2007 Ottawa, Canada, June 20-22, 2007, Revised Selected Papers*, pages 62–76, 2007.

APPENDIX A
BETTERTIMES EXTENSION

This section describes a small extension to the BetterTimes system. The extension allows for intermediate values to be securely used outside of the circuit. The *assurance value*, which is an internal value of a BetterTimes formula, is handled separately for these cases. The assurance value carries evidence of whether or not *Alice* has cheated up to the point when the assurance value is computed. The assurance value of *all* outputs are combined using the extension, such that misbehavior at any time while the formula is computed yields only $\perp$ values. Figure 5 shows an example of how the extension is used in DecentMP.

Some background about the features which are used by BetterTimes-instructions are needed in the context of this new construction. A BetterTimes formula is composed of instructions, these arithmetic instructions are called BetterTimes-instructions. BetterTimes-instructions are a recursive data structure, where an instantiated instruction is either a binary operation, for which each operand is another instruction, or a scalar value. These are an addition function $[\![c_1]\!] \oplus [\![c_2]\!] = E(m_1 + m_2)$,

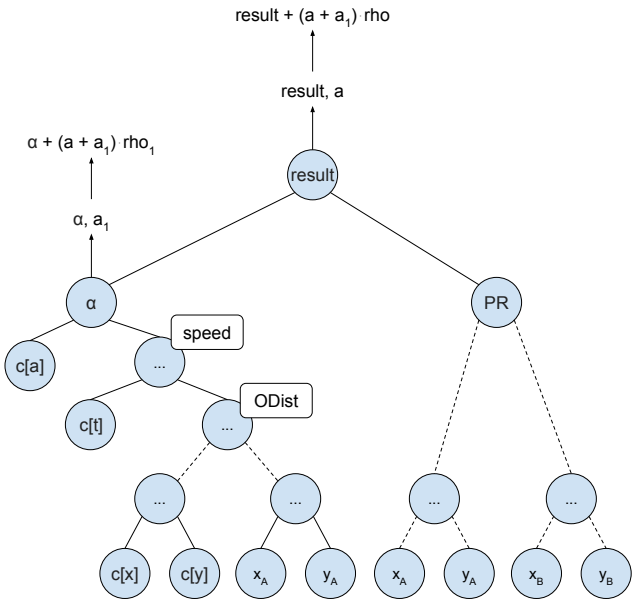

Fig. 5: Example of how the new BetterTimes extension is used

a unary negation function $\neg[\![c_1]\!] = E(-m_1)$ and a multiplication function $[\![c_1]\!] \odot m_2 = E(m_1 \cdot m_2)$. Where $[\![x]\!]$ is used to denote that the variable x is encrypted using *Alice*'s public key, $E(m)$ is the encryption function applied to m using *Alice*'s public key, and $[\![c_1]\!]$ and $[\![c_2]\!]$ encrypts m_1 and m_2, respectively BetterTimes also provide means of computing multiplications of two ciphertexts as $[\![c_1]\!] \odot [\![c_2]\!] = E(m_1 \cdot m_2)$ through a secure outsourcing technique in which *Alice* is engaged in interactive protocol.

Definition 10 (Intermediate BetterTimes outputs)**.** *The new BetterTimes operation Ins_O registers the result of an instruction as intermediate output.*

As defined in the original paper $evaluate$ computes the formula, and accumulates all assurance values $[\![a_i]\!]$. The assurance values are summed as the evaluation proceeds. Finally, the assurance value is randomized and added to the result as $([\![result]\!] \oplus [\![a]\!]) \odot \rho$, with ρ random. Note that these operations do not take place in the plaintexts, but are computed homomorphically on the ciphertext. Using the extension, when $evaluate$ encounters an Ins_O instruction, it computes the result and the assurance value as normal, but also saves the intermediate result $[\![z_i]\!]$ and the current assurance value $[\![a_i]\!]$ as a tuple $([\![z_i]\!], [\![a_i]\!])$ to a store S.

After a formula has been fully evaluated, instead of outputting the result directly, a list is constructed where all outputs are assured using a overarching assurance value A, defined as:

$$[\![A]\!] = [\![a]\!] \oplus \left(\bigoplus_{i=0}^{|S|} snd(S[i]) \right)$$

Where $[\![a]\!]$ is the final assurance value and $snd()$ retrieves the second value in a tuple ($fst()$ is used below to retrieve the first value).

Finally, the array of the final result and all intermediate values is given as:

$$[([\![result]\!] \oplus [\![A]\!] \odot \rho] :: [(fst(s) \oplus [\![A]\!]) \odot \rho_i \text{ for } s \in S]$$

Where ρ and all ρ_i are independent and uniformly random variables, and :: denotes array concatenation.

APPENDIX B
SMC NOTIONS

In the following, recall briefly some fundamental concepts from SMC.

A. Negligible Functions

For the scope of this work, what it means for a function to be negligible is shown in Definition 11

Definition 11. *A function $\epsilon : \mathbb{N} \to \mathbb{R}$ is said to be 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.

B. Indistinguishability

Indistinguishability is an important notion for the proofs presented in this paper, and is specified formally in Definition 12.

Definition 12. *The two random variables $X(n, a)$ and $Y(n, a)$ (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 a probabilistic polynomial time (PPT) adversary $\mathcal{A}$ the following function is negligible:*

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

C. Ideal and Real Execution

The IDEAL and REAL executions follow the definitions of Pinkas and Lindell [16]. As highlighted previously, in the IDEAL model the parties interact only with a trusted third party, which ensures that the executed protocol matches exactly an implementation of the functionality, where parties cannot deviate from the protocol. In the REAL model, instead a concrete instance of the protocol is considered.

Let $\overrightarrow{x} \in I_A$ and $\overrightarrow{y} \in I_B$ be the private inputs for two parties, and let $g(\overrightarrow{x}, \overrightarrow{y}) \in O_A \times O_B$ be the output of a functionality g.

Here follows a brief recap of the execution in the IDEAL model, where an adversary $\mathcal{A}_{\text{IDEAL}}$ is controlling a corrupted party (Alice for the context of this paper). The benign party (Bob) sends their input $\overrightarrow{y}$ to the trusted party, and $\mathcal{A}_{\text{IDEAL}}$ tells the corrupted party (Alice) to either send the actual input of Alice (which can be read by $\mathcal{A}_{\text{IDEAL}}$) or another value of the same length as $\overrightarrow{x}$ to the trusted party. The trusted party then computes the output to be received by both parties. For the scope of this paper, Bob has no output, and Alice receives $g(\overrightarrow{x}, \overrightarrow{y})$. Alice forwards the result to $\mathcal{A}_{\text{IDEAL}}$. The original definition also handles the case when $\mathcal{A}_{\text{IDEAL}}$ wishes to abort the protocol. In the context of this paper, since the SMC solution is based on homomorphic encryption, *Bob* receives no output from the ideal functionality. Therefore, it does not make sense for the adversary to abort the protocol. Also, this means that fairness guarantees for *Bob* are out of scope, so abortions of the protocol by the simulator do not need to be accounted for.

After the *ideal execution* of a functionality on inputs $(\overrightarrow{x}, \overrightarrow{y})$ $\mathcal{A}$ outputs an arbitrary PPT function on the private input of *Alice* and the output of the functionality $(\overrightarrow{x}, g(\overrightarrow{x}, \overrightarrow{y}))$. Formally thus:

$$\mathcal{A}_{\text{IDEAL}} : I_A \times O_A \to O_{\mathcal{A}}$$

for an arbitrary but fixed output space $O_{\mathcal{A}}$ (for instance a string of bits of length n).

The *real execution* of a concrete protocol π is rather intuitive, where $\mathcal{A}_{\text{REAL}}$ takes the place of the corrupted party and acts on their behalf. In this case:

$$\mathcal{A}_{\text{REAL}} : I_A \times view_\pi^A \times O_A \to O_{\mathcal{A}}$$

where $view_\pi^A$ are the intermediate values seen by $\mathcal{A}_{\text{REAL}}$ during the execution of π.

Recall that Definition 8 requires that for any adversary $\mathcal{A}_{\text{REAL}}$ against a protocol, there exists a simulator:

$$\mathcal{S} : I_A \times O_A \to O_{\mathcal{A}}$$

such that the distribution of the outputs of $\mathcal{S}$ and $\mathcal{A}$ are computationally indistinguishable:

$$\{\text{IDEAL}_{g,\mathcal{S}}(\overrightarrow{x}, \overrightarrow{y})\} \overset{c}{\equiv} \{\text{REAL}_{\pi,\mathcal{A}}(\overrightarrow{x}, \overrightarrow{y})\}$$

That is, a protocol π is privacy-preserving if it is possible to construct a concrete PPT $\mathcal{S}$ such that for every attacker $\mathcal{A}$ against the real protocol, using only the information available to the attacker by construction (their inputs and the output), its output is indistinguishable from the one of $\mathcal{A}$. If this is possible, an adversary does not learn anything apart from what is disclosed by the functionality by attacking π.

APPENDIX C
PROOFS

Recall that by definition, BetterTimes outsources multiplications to *Alice* as depicted in Fig. 6. Since algorithm is probabilistic, we use the pWhile probabilistic imperative programming language of [1] for the description, where $x \xleftarrow{\$} M$ stands for uniformly random assignation of a value in the set M to x.

Proof of Lemma 1. It follows from the definition of Figure 6, that the intermediate values observed by an adversary $\mathcal{A}$ during the protocol execution are encryptions of uniformly random independent triples $(\perp, \perp, \perp)$ corresponding to $[\![x']\!], [\![y']\!]$ and $[\![c]\!]$, which are independently blinded.

From Lemma 1 of [10], the decryption of the assurance value $[\![a]\!]$ is indistinguishable from $\perp$ if an adversary is dishonest. Since the assurance value is added to the result of the computation, the final value is also $\perp$.

Proc. $BetterTimes([\![x]\!], [\![y]\!])$:
$c_a \xleftarrow{\$} \{0..p\}; c_m \xleftarrow{\$} \{1..p\};$
$b_x \xleftarrow{\$} \{0..p\}; b_y \xleftarrow{\$} \{0..p\};$
$\rho \xleftarrow{\$} \{1..p\};$
// Blind operands
$[\![x']\!] \leftarrow [\![x]\!] \oplus [\![b_x]\!]; [\![y']\!] \leftarrow [\![y]\!] \oplus [\![b_y]\!];$
// Create challenge
$[\![c]\!] \leftarrow ([\![x']\!] \oplus [\![c_a]\!]) \odot c_m;$
// Outsource multiplication
$([\![z']\!], [\![a']\!]) \leftarrow OS([\![x']\!], [\![y']\!], [\![c]\!]);$
// Compute assurance value
$[\![a]\!] \leftarrow ([\![a']\!] \ominus [\![z']\!] \odot c_m \ominus [\![y']\!] \odot (c_a \cdot c_m)) \odot \rho;$
// Un-blind multiplication
$[\![z]\!] \leftarrow [\![z']\!] \ominus ([\![x']\!] \odot b_y \oplus [\![y']\!] \odot b_x \oplus [\![b_x \cdot b_y]\!])$;
return $([\![a]\!], [\![z]\!])$;

Fig. 6: Outsourced multiplication with BetterTimes

□

Now for the proof of Lemma 2:

Proof of Lemma 2. By construction, all intermediate messages remain unchanged by the introduction of Ins_O. Showing that intermediate messages and the final output are $\perp$ if *Alice* is dishonest follows from the observations in the proof of Lemma 1, with one small addition. The sum $[\![a]\!] + \bigoplus_{i=0}^{|S|} snd(S[i])$ is the encryption of 0 only with negligible probability since all terms are fresh uniformly random variables. □

Appendix D
EasyTimes: syntactic improvement of BetterTimes

Herein a python-like syntax is described, for which a direct translation into BetterTimes syntax is provided. The code presented here is python-compatible, and thus can be executed by the regular python interpreter. See Listing 6 for an example of how this is applied to a concrete example.

Listing 6: Example evaluation

```
evaluator = StringEvaluator()
formula = SecureFormula(evaluator, 4, 3, 2, 1)

def foo(x, z, i2):
  return x + z - i2

with formula as sf:
  i1, i2, i3, i4 = sf.inputs

  c = i1 + i2
  sf.output(c)

  z = i4
  x = i1 * 42 + i2
  x = foo(x, z, i2)
  y = i2 * i3 + i4
  o_1 = x + y
  o_2 = y * z

  sf.output(c * (i3 - i2))

outs = formula.evaluate()
for out in outs:
  print(out)
```

In Listing 6, there are four inputs provided to the formula. There are two outputs, $i_1 + i_2$ and finally $x * y$, which is computed using a separate method. Regardless of how the output is constructed, even when control flow primitives such as if statements and loops are used, the result is a single BetterTimes formula. A `SecureFormula` requires an *evaluator* (which will be discussed later) and an arbitrary number of inputs. The operations done with the inputs, and what the resulting outputs are, is defined by operating inside a **`with`** block. Upon leaving the **`with`** block, variables are no longer mutable, and the formula is fixed. Thus, at this point, the formula can be evaluated. An implementation of such a construct is concise in python, as shown in Listing 7.

Listing 7: Python-class for secure formula

```
class SecureFormula(object):
  def __init__(self, evaluator, *inputs):
    self.__evaluator = evaluator
    formula_inputs = []
    for inp in inputs:
      formula_inputs.append(
          FormulaInstruction(inp))

    self.__context = FormulaContext(
        formula_inputs)
    self.__outputs = None

  def __enter__(self):
    return self.__context

  def __exit__(self, exc_type, exc_val,
               exc_tb):
    self.__outputs = self.__context.outputs

  def evaluate(self):
    return [self.__evaluator.evaluate(out)
            for out in self.__outputs]
```

The `__enter__` method returns a `FormulaContext` object, and is implicitly called when a `SecureFormula` instance is used for a **`with`** statement. The context is basically just a controlled store for inputs and outputs, as seen in Listing 8.

Listing 8: Python-class for formula context

```
class FormulaContext(object):
  def __init__(self, inputs):
    self.__inputs = inputs
    self.__outputs = []
```

```
  def output(self, output):
    self.__outputs.append(output)

  @property
  def inputs(self):
    return self.__inputs

  @property
  def outputs(self):
    return self.__outputs
```

From the context, representations of the input variables can be obtained. When a variable is provided to the `sf.output()` method, the result is marked as a part of the output of the context. representing are stored, by the construction of the `FormulaInstruction` object. A slightly simplified `FormulaInstruction` is seen in Listing 9.

Listing 9: Python-class for formula instruction

```
class FormulaInstruction(object):
  __ADD = object()
  __SUB = object()
  __MUL = object()

  def __init__(self, a=None, b=None, c=None):
    if b:
      self._operation = a
      self._left_operand = b
      self._right_operand = c
    else:
      self._scalar = a

  @property
  def left_operand(self):
    return self._left_operand

  @property
  def right_operand(self):
    return self._right_operand

  @property
  def scalar(self):
    if self.is_scalar():
      return self._scalar
    else:
      return None

  def is_scalar(self):
    return hasattr(self, '_scalar')

  def is_add(self):
    return self._operation == self.__ADD

  def is_sub(self):
    return self._operation == self.__SUB

  def is_mul(self):
    return self._operation == self.__MUL

  def __op(self, op, o1, o2):
    return FormulaInstruction(op, o1, o2)

  def __add__(self, other):
    return self.__op(self.__ADD, self, other)

  def __sub__(self, other):
    return self.__op(self.__SUB, self, other)

  def __mul__(self, other):
    return self.__op(self.__MUL, self, other)
```

Now to see how accumulating `FormulaInstruction`'s and storing them as outputs is a mapping to BetterTimes instructions, consider the following `StringEvaluator` shown in Listing 10. This evaluator consumes an instruction, recursively explores all branches of the formula, and prints them in BetterTimes syntax.

Listing 10: Python-class for formula instruction

```
class StringEvaluator(FormulaEvaluator):
  def evaluate(self, instruction):
    if instruction.is_scalar():
      return 'Ins(%s)' % instruction.scalar
    return "Ins(%s, %s, %s)" % (
      self.__bt_op(instruction),
      self.evaluate(instruction.left_operand),
      self.evaluate(instruction.right_operand)
    )

  def __bt_op(a, instruction):
    if instruction.is_add():
      return 'ADD'
    elif instruction.is_sub():
      return 'SUB'
    elif instruction.is_mul():
      return 'MUL'
```

By mapping an arbitrary `SecureFormula` to BetterTimes syntax, it's easy to see that any formula constructed using this syntax can be evaluated securely using the BetterTimes system.