Outsourcing MPC Precomputation for Location Privacy

*** Accepted for publication at EuroS&P Location Privacy Workshop 2022. This pdf will be updated with the IEEE copyright notice and DOI as soon as it's published.**

Ivan Oleynikov
Computer Science and Engineering
Chalmers University
Gothenburg, Sweden
ivanol@chalmers.se

Elena Pagnin
Elec. And Info. Technology
Lund University
Lund, Sweden
elena.pagnin@eit.lth.se

Andrei Sabelfeld
Computer Science and Engineering
Chalmers University
Gothenburg, Sweden
andrei@chalmers.se

***Abstract*—Proximity testing is at the core of several Location-Based Services (LBS) offered by, e.g., Uber, Facebook, and BlaBlaCar, as it determines closeness to a target. Unfortunately, modern LBS demand not only that clients disclose their locations in plain, but also to trust that the services will not abuse this information. These requirements are unfounded as there are ways to perform proximity testing without revealing one's location.**

We propose POLAR, a protocol that implements privacy-preserving proximity testing for LBS. POLAR is suitable for clients running mobile devices, and relies on a careful combination of three well-established multiparty computation protocols and lightweight cryptography. A point of originality is the inclusion of two servers into the proximity testing. The servers may aid multiple pairs of clients and contribute towards enhancing privacy, improving efficiency, and reducing the running time of clients' procedures.

***Index Terms*—privacy, proximity-testing, multiparty computation, active security**

1. Introduction

Location-Based Services (LBS) are any services that rely on user location data to operate. For example, a mapping service needs a client's location to suggest the relevant places nearby, a taxi service needs a client's location to match it with the nearest drivers, smart home devices may use the owner's location to turn on light, music, and air conditioning when they enter their home.

Proximity testing (PT) is the problem of deciding whether a two or more inputs lie within a certain distance of one another. This paper focuses on *location proximity testing* by means of privacy-enhancing protocols operating on user locations. There exist approaches to PT that implement it by means of direct communication and measuring signal strength using, e.g., Bluetooth [5], [40] (adopted in some COVID-19 contact tracing apps). While these solutions might provide an accurate distance calculation, our approach occupies a different niche: indeed, in some LBS it might not be possible for users to pick each others' signals (e.g., planning for a shared ride between towns; matching with proximity radius larger than the signal range; or matching with offline users).

Modern ridesharing and taxi services match drivers and passengers according to the proximity of their routes, or of the start and endpoints of their journeys so that the passenger can easily walk the remaining distance to reach the desired destination. Messaging and dating apps use proximity testing to match users who are in the same area, and online mapping services use it to help users discover close-by places (e.g. coffee shops, supermarkets). Geofencing apps for children, pets, vehicles, and vessels also draw on proximity testing.

In current practice, the widely adopted LBS are centralized and require full-trust from the clients in order to deliver the desired functionality. A client is expected to reveal their location to the service; and clients cannot check if their data has been used the way they expect, whether it has been misused by the LBS provider or stolen by an attacker who breached the security of LBS provider. For example, Snapchat employees reportedly abused their privileges to spy on users' location data [7], and similar cases were reported about Uber [20], Yahoo [6] and Facebook [8]. This raises privacy concerns over the existing practices and motivates the search for solutions that would ensure the privacy of user data.

This paper considers the problem of constructing a cryptographic protocol that performs PT in a privacy-enhancing way. A privacy-enhancing PT protocol is required to be correct (provide the right answer); and secure (preserve input privacy) by revealing only the outcome of the proximity test, and no further information about users' concrete locations. In the remainder of the paper, whenever we refer to proximity testing, we will mean privacy-enhancing proximity testing.

Formalizing Proximity Testing. There exist multiple approaches to formalizing what location is and what is proximity in proximity testing. The grid-based approach [13], [26], [28], [29], [30], [38], [39], [41] divides the whole location space into a grid of cells, the clients determine the cell they are in and then

simply perform equality test on their cell identifiers. Although it might be tempting to do proximity testing at a cost of a simple equality test, this approach suffers from inherent imprecision since the clients are matched only if they are in the same grid cell. Another alternative is polygon-based matching [23], which is also imprecise and becomes less efficient if one wants to approximate a circle with a complex polygon. We follow the line of work on Euclidean distance based matching [18], [23], [31], [32], because it is precise and it naturally arises in some important applications, e.g., messengers and social networks suggesting friends nearby to a user, geofencing. Euclidean distance may serve as an approximation of other distance measures like road distance on a given map or Manhattan distance.

In this work, we consider users' locations to be points on a (discretized) Euclidean plane (which can be used to approximate distance on a small enough region of Earth surface). Our functionality matches two users (outputs 1 instead of 0) if the distance between their input locations does not exceed a threshold radius value R, on which they agree beforehand. The threshold radius R here serves as a parameter of the protocol, and can be chosen to be any positive integer when instantiating the protocol; it is fixed and public, i.e. known to all the parties prior to protocol start. We focus on the case of 2-dimensional client locations (i.e. belonging to a Euclidean plane) for a fair comparison with prior work, but it is not essential for our protocol which easily generalizes to n-dimensional Euclidean distance based matching.

Tools we use. In a nutshell, the protocols we consider here are expected to compute the function (where (x_A, y_A) comes from Alice, (x_B, y_B) from Bob, and the result 0 or 1 goes to Alice):

$$f((x_A, y_A), (x_B, y_B)) = \begin{cases} 1, & \text{if } (x_A - x_B)^2 + (y_A - y_B)^2 \leq R^2, \\ 0, & \text{otherwise.} \end{cases} \quad (1)$$

Secure Multi-Party Computation (MPC) protocols allow parties to secret-share some values between each other, do some operations on the shared values and then reconstruct the shares of the result. MPC essentially implements a secure, trusted virtual machine that accepts some values on input, performs computations and returns the result. Actively secure MPC protocols stay secure even in the presence of malicious adversaries that may change the behavior of corrupted parties that participate in the computation.

Actively secure MPC is as a natural tool to implement functionalities that involve relatively few operations, like the one in Equation (1). Until now, the high demands imposed by actively secure MPC on computation and communication have prevented it from being applied to efficient and scalable PT solutions. In this work, we finally bypass this obstacle by leveraging the fact that many MPC protocols run in two phases: first heavy precomputation phase (which is often too heavy for resource-constrained mobile devices) that does not depend on the actual inputs of the parties, but merely generates correlated randomness; and, secondly, a relatively fast online phase that uses the correlated randomness from the previous phase and the party inputs to compute the final output. Our approach is to include two servers in the PT and offload to these all precomputations run in the first phase. This way, the heaviest burden is done by the servers allowing clients to run only the light online phase. The privacy cost of this speedup is moderate: even though the servers are involved in the protocol, they never get to handle any data that is derived from client inputs and therefore the impact of corrupted servers is limited. It is important to note that in our proposed technique none of the servers can see or modify the precomputed data that the two of them are generating: the servers run another MPC protocol between them that produces the precomputed data, and then they transfer it properly masked and authenticated to the clients without ever seeing the data themselves (assuming that only one of the two servers is corrupted, and the other one is honest).

In other words, compared to the server-less setting of the previous protocols [18], [23], [32], our setting allows better client performance at the cost of introducing an extra assumption that at least one of the servers must be honest. Our setting has an additional practical benefit: a single pair of servers can supply multiple pairs of clients with precomputed data, and generate the data in bigger batches more efficiently (the amortized cost of precomputation phase usually goes down if one increases the amount of precomputation that must be done in one run). This can be thought of as an analogue of the economy of scale: the more clients you serve, the less computation you have to pay with for each client pair.

Albeit we demonstrate the application of this two-server aided precomputation setting by applying it to location privacy problem, we argue it may be useful in other fields as well. Any setting where resource-constrained clients want to run an actively secure MPC protocol with few operations in it may benefit from augmenting it with two servers to outsource the precomputation phase to. For example, such setting can naturally arise in a housing rental app, phonebook contact discovery (used by messengers), collaborative planning apps (which may have more than two users).

Overview of our protocol. In our proposed POLAR (Precomputation fOr LocAtion pRivacy) protocol, the clients run a combination of Tinier [12], SPDZ [9] and edaBits [11] in order to compute the proximity testing functionality. And the servers use the same combination of protocols (Tinier + SPDZ + edaBits), in order to create the precomputation data for the clients. All four parties use the outsourcing technique of [21] to transfer the precomputed data from the servers' MPC protocol to the clients without revealing it to any of the two servers.

Our contribution. This paper presents POLAR, a novel, actively secure protocol for server-aided privacy-preserving location proximity testing. In detail, we provide a formal description of the POLAR protocol and its building blocks. We formally prove its security in Canetti's hybrid model [4]. In addi-

tion, we develop a proof of concept implementation of POLAR and compare its performance against OLIC [31] which also uses two servers (but for a different purpose), $\text{ABY}^{\text{C}}_{\text{Y}}$ and $\text{ABY}^{\text{C}}_{\text{AY}}$ [23] which work in the server-less setting. Given that POLAR is the first server-precomputation aided PT protocol, there is no clear competitor for comparison. We thus compare against server-less and server-aided protocols to see how much of a performance improvement can one gain by offloading the precomputation to the servers.

Our evaluations show that the client cost of running POLAR is negligible (around 10 ms CPU time and under 2 KB communication). While the resources required from the servers are quite moderate: below 0.5 seconds CPU time and 84 MB communication per run when amortized over 2000 runs. 84 MB per client pair is an acceptable amount of communication for servers, it is equivalent to streaming a short video. When compared to other protocols, POLAR's main advantages are active security and unmatched client performance (around 2 ms of CPU time and 2 KB of total communication). The server performance of POLAR also beats the server performance of OLIC for large enough values of radius R (the performance of OLIC depends on R), but still stays a lot heavier than the client performance of $\text{ABY}^{\text{C}}_{\text{AY}}$ and $\text{ABY}^{\text{C}}_{\text{Y}}$.

2. Preliminaries

The clients in POLAR run a combination of MPC protocols in order to implement the PT functionality (Equation 1). In the following, we explain what this combination cosists of and how we model it using a blackbox idea functionality.

MPC protocols are usually limited to operations in one specific domain. Arithmetic MPC protocols work with integers modulo some value (prime number p in our case), binary MPC protocols work with bits. Arithmetic domain is particularly good for computations that use a lot of numeric additions and multiplications, while the binary domain can represent numbers in their bit-decomposed form and cheaply evaluate Boolean circuits on them (e.g. comparison, bit shifts, sorting, indexing). The function that clients want to compute (shown on Equation 1) in our protocol tends to mix the two types of operations: on one hand, it has additions and multiplications, on the other hand, it has comparison. Therefore, to evaluate it efficiently we combine two MPC protocols to work over both domains and we use edaBits [11] technique to convert the values between two domains.

Ideal Functionality. To model the mixed arithmetic-binary MPC, we make black-box usage of the functionality $\mathcal{F}_{\text{AB-MPC}}$ shown on Figure 1. This functionality can be implemented by the edaBits [11] technique. Most of the commands in $\mathcal{F}_{\text{AB-MPC}}$ repeat the functionality on which the edaBits is built, except for the commands `ConvertA2B` and `Compare` which are implemented using the edaBits technique itself. The `Compare` is obtained by combining the other commands of $\mathcal{F}_{\text{AB-MPC}}$, but there are multiple ways to do that (e.g., using a Boolean comparison circuit or with probabilistic truncation [11]). For the sake of generality, we define `Compare` as a standalone command and leave its specification up to specific implementations. The $\mathcal{F}_{\text{AB-MPC}}$ functionality is implemented by MP-SPDZ [24] framework (which we use for our benchmarks) using the techniques of daBits [34] and edaBits [11].

Notation We will use the notation $[\![x]\!]_p$ for value $x \in \mathbb{Z}_p$ being input into the $\mathcal{F}_{\text{AB-MPC}}$ with type = `arithmetic`, and $[\![x]\!]_2$ for value $x \in \{0,1\}$ with type = `binary` (the variable names x are assumed to be unique over both `arithmetic` and `binary` domains). When describing protocols that use $\mathcal{F}_{\text{AB-MPC}}$ in pseudocode, we will use the listed message types as procedure names, e.g., $[\![x]\!]_p \leftarrow \texttt{ConvertB2A}([\![y]\!]_2)$ means sending $(\texttt{ConvertB2A}, \text{“}x\text{”}, \text{“}y\text{”})$ to the $\mathcal{F}_{\text{AB-MPC}}$. We will also use values $[\![\cdot]\!]_p$ in arithmetic expressions and $[\![\cdot]\!]_2$ in Boolean expressions (i.e., arithmetics over $\mathbb{F}_2$), implying evaluation of the corresponding expressions using `Mult` and `LinComb`. For a vector of bits $v = (v_0, \dots v_{k-1})$ we will write $[\![\overrightarrow{v}]\!]_2$ to denote a vector of bits $([\![v_0]\!]_2, \dots [\![v_{k-1}]\!]_2)$, all of which are in the binary domain of $\mathcal{F}_{\text{AB-MPC}}$.

For example, consider the Boolean inner product function $\text{IP}(u,v) = \sum_{i=0}^{k-1} u_i v_i = \bigoplus_{i=0}^{k-1} u_i \wedge v_i$. If we have the vectors u and v input into the binary domain of $\mathcal{F}_{\text{AB-MPC}}$ as $[\![\overrightarrow{u}]\!]_2 = ([\![u_0]\!]_2, \dots [\![u_{k-1}]\!]_2)$ and $[\![\overrightarrow{v}]\!]_2 = ([\![v_0]\!]_2, \dots [\![v_{k-1}]\!]_2)$, we can write $[\![b]\!]_2 \leftarrow \text{IP}([\![\overrightarrow{u}]\!]_2, [\![\overrightarrow{v}]\!]_2)$ to denote the computation of inner product inside $\mathcal{F}_{\text{AB-MPC}}$ via the operations

$$[\![p_0]\!]_2 \leftarrow \texttt{Mult}(\texttt{binary}, [\![u_0]\!]_2, [\![v_0]\!]_2)$$
$$\dots$$
$$[\![p_{k-1}]\!]_2 \leftarrow \texttt{Mult}(\texttt{binary}, [\![u_{k-1}]\!]_2, [\![v_{k-1}]\!]_2)$$
$$c \leftarrow (0, 1, \dots, 1) \in \mathbb{F}_2^{k+1}$$
(The next line is computing the sum of all $[\![p_i]\!]_2$)
$$[\![b]\!]_2 \leftarrow \texttt{LinComb}(\texttt{binary}, [\![\overrightarrow{p}]\!]_2, c).$$

In real-life, the $\mathcal{F}_{\text{AB-MPC}}$ will be implemented by a combination of edaBits, SDPZ [9] and Tinier [12]. Note that all three techniques rely on preprocessing data (correlated random values held by the parties) to operate. When the clients implement $\mathcal{F}_{\text{AB-MPC}}$, they will use preprocessing data generated for them by the servers. The servers will generate the data using an MPC protocol and then transfer it to the clients without any of the two servers being able to see or modify the data (as long as the other server is honest).

The servers' MPC that generates and transfers the precomputation data to the clients is modelled by $\mathcal{F}_{\text{Out-MPC}}$ ideal functionality shown on Figure 2. The precomputed data for the clients generated by this functionality consists of SPDZ [9] multiplication triples, Tinier [12] multiplication triples, random Tinier shares (to allow the clients to input values into Tinier), daBits [34] and edaBits [11]. In practice, the $\mathcal{F}_{\text{Out-MPC}}$ functionality will be implemented by applying the outsourcing technique [21] to the combination of edaBits, SPDZ and Tinier (distinct instances, not the ones used by clients). The servers, unlike clients, generate their own preprocessing data using the relatively expensive precomputation protocols. We do not show how $\mathcal{F}_{\text{Out-MPC}}$ is implemented in detail,

Setting: the ideal setting consists of $\mathcal{F}_{\text{AB-MPC}}$ functionality and the parties P_1 ...P_n using it.

Input: On input $(\texttt{Input}, P_i, \text{type}, \text{id}, x)$ from P_i and $(\texttt{Input}, P_i, \text{type}, \text{id})$ from all other parties, with id a fresh identifier, type $\in \{\texttt{binary}, \texttt{arithmetic}\}$ and $x \in \mathbb{Z}_2$ or $x \in \mathbb{Z}_p$ (depending on type), store $(\text{type}, \text{id}, x)$.

Linear Combination: On input $(\texttt{LinComb}, \text{type}, \text{id}, (\text{id}_i)_{i=1}^m, (c_j)_{j=0}^m)$, where each id_j is stored in memory and $c_j \in \mathbb{Z}_2$ if type $=$ `binary` or $c_j \in \mathbb{Z}_p$ if type $=$ `arithmetic`, retrieve $((\text{type}, \text{id}_1, x_1), \dots (\text{type}, \text{id}_m, x_m))$, compute $y = c_0 + \sum_{i=1}^m x_i \cdot c_i$ modulo 2 if type $=$ `binary` and modulo p if type $=$ `arithmetic`, and store $(\text{type}, \text{id}, y)$.

Multiply: On input $(\texttt{Mult}, \text{type}, \text{id}, \text{id}_1, \text{id}_2)$ from all parties (where id_1, id_2 are present in memory), retrieve $(\text{type}, \text{id}_1, x)$, $(\text{type}, \text{id}_2, y)$, compute $z = x \cdot y$ modulo 2 if type $=$ `binary` and modulo p if type $=$ `arithmetic`, and store (id, z).

From Binary to Arithmetic: On input $(\texttt{ConvertB2A}, \text{id}, \text{id}')$ from all parties, retrieve $(\texttt{binary}, \text{id}', x)$ and store $(\texttt{arithmetic}, \text{id}, x)$.

From Arithmetic to Binary: On input $(\texttt{ConvertA2B}, \text{id}_0 \dots \text{id}_{l-1}, \text{id}')$ from all parties, retrieve $(\texttt{arithmetic}, \text{id}', x)$, bit-decompose it into $(x_0, \dots x_{k-1})$ and store $((\texttt{binary}, \text{id}_0, x_0), \dots (\texttt{binary}, \text{id}_{l-1}, x_{l-1}))$.

Compare: On input $(\texttt{Compare}, \text{id}, \text{id}', y)$ from all parties, where $y \in \mathbb{Z}_p$, retrieve $(\texttt{arithmetic}, \text{id}, x)$, store $(\texttt{binary}, \text{id}', 1)$ if $x \leq y$ or $(\texttt{binary}, \text{id}', 0)$ otherwise.

Output: On input $(\texttt{Output}, \text{type}, \text{id})$ from all honest parties (where id is present in memory), retrieve $(\text{type}, \text{id}, y)$ and output it to the adversary. Wait for an input from the adversary; if this is `Deliver` then output y to all parties, otherwise output `Abort`.

Figure 1: Ideal functionality $\mathcal{F}_{\text{AB-MPC}}$ of MPC arithmetic blackbox modulo 2 and modulo p [11]

because it is trivial and unnecessarily technical for our presentation. Full details can be found in the source code of our implementation.

This functionality works with two servers `Server-1` and `Server-2`, and two clients `Alice` and `Bob`.

Eval. On command `Eval` from both servers, generate the precomputation data for the clients. Save `Alice`'s data as $z_{\texttt{Alice}}$ and `Bob`'s data as $z_{\texttt{Bob}}$. Output `Eval` to the adversary. If `Alice` and `Bob` are corrupted, deliver the corresponding $z_{\cdot}$ to the corrupted clients.

Output. On command $(\texttt{Deliver}, C)$ from the adversary where C is either `Alice` or `Bob`, deliver z_C to the corresponding client.

Figure 2: Functionality $\mathcal{F}_{\text{Out-MPC}}$ for outsourced evaluation of precomputation data for $\mathcal{F}_{\text{AB-MPC}}$. It is implemented by the outsourcing technique [21, Figure 3].

3. The POLAR Protocol

Formally, we describe our protocol in the $(\mathcal{F}_{\text{Out-MPC}}, \mathcal{F}_{\text{AB-MPC}})$-hybrid model [27]. We model it as an interaction of clients Alice and Bob, the two servers and the ideal functionalities $\mathcal{F}_{\text{Out-MPC}}$ (Figure 2) and $\mathcal{F}_{\text{AB-MPC}}$ (Figure 1). In practice, the two ideal functionalities will be replaced by the corresponding protocols that implement them [11], [21] and yield a protocol that implements our functionality on four parties. The $\mathcal{F}_{\text{AB-MPC}}$ functionality allows Alice and Bob to compute the function they want (Equation 1). But implementing $\mathcal{F}_{\text{AB-MPC}}$ directly would be too costly for them, therefore they also use the $\mathcal{F}_{\text{Out-MPC}}$ together with the servers which computes the precomputation data (needed for $\mathcal{F}_{\text{AB-MPC}}$) and delivers it to the clients (without revealing it to the servers). In practice, the $\mathcal{F}_{\text{Out-MPC}}$ will be implemented via an MPC protocol (similar to $\mathcal{F}_{\text{AB-MPC}}$, but with different parameters; we do not focus on it much here) ran by the servers which delivers its result to the clients; while $\mathcal{F}_{\text{AB-MPC}}$ will be implemented by an MPC protocol ran by the clients that uses the precomputation data from $\mathcal{F}_{\text{AB-MPC}}$ to ease the computation and communication load of the clients.

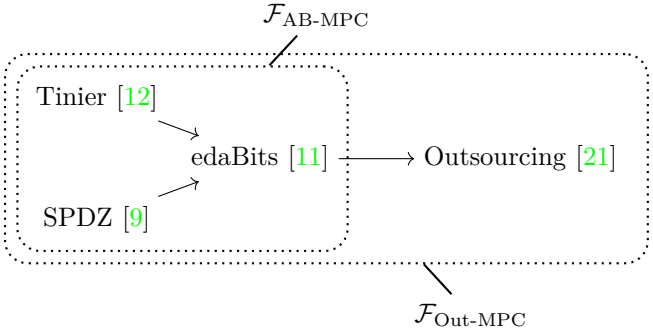


Figure 3: The diagram of blackbox applications of previous works that yields the functionalities that we use in our hybrid model.

Figure 3 gives an overview of the order in which the existing techniques are applied to one another by the clients in order to obtain the ideal functionalities that we use. Here, Tinier prodives MPC computations in the binary domain, SPDZ provides computations in arithmetic domain, edaBits combines the two to implement a single MPC capable of doing both and converting between them, and, finally, the outsourcing technique allows the clients to securely receive their precomputed data from the edaBits MPC even if one of the servers is untrusted. In the following, we give an overview of how the POLAR protocol works.

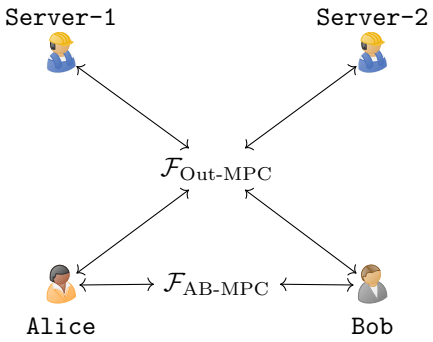


Figure 4: Diagram of the hybrid setting we describe our protocol POLAR in.

POLAR involves four parties (Figure 4): two servers `Server-1` and `Server-2`; and two clients `Alice` and `Bob`. They also have access to the mentioned $\mathcal{F}_{\text{AB-MPC}}$ (only for `Alice` and `Bob`) and $\mathcal{F}_{\text{Out-MPC}}$ functionalities. `Alice` and `Bob` know their respective

locations (x_a, y_a) and (x_b, y_b). At the end of the protocol execution, `Alice` gets a bit ρ; $\rho = 1$ if her distance to `Bob` is less than or equal to a given public value R, otherwise $\rho = 0$. Figure 5 shows the formal definition of the ideal functionality $\mathcal{F}_{\text{PT}}$ that POLAR implements, while Figure 6 shows how POLAR implements $\mathcal{F}_{\text{PT}}$ in our hybrid setting.

Parameters: a positive number R, the radius of proximity testing; k, the bit width of clients' coordinates.
Setup: Four parties, `Alice`, `Bob`, `Server-1`, `Server-2`. `Alice` and `Bob` hold inputs $(x_a, y_a) \in \mathbb{Z}_p^2$ and $(x_b, y_b) \in \mathbb{Z}_p^2$ respectively

1) Receive (x_a, y_a) from `Alice`, and (x_b, y_b) from `Bob`. Ensure that each value x_a, y_a, x_b, y_b consists of exactly k bits; if not, abort.
2) Receive `Deliver` from both servers. If one of them sends something else, abort.
3) Send $\rho = 1$ to `Alice` if $(x_a - x_b)^2 + (y_a - y_b)^2 \leq R^2$, and $\rho = 0$ otherwise.
4) Send `Received` to both servers.

Figure 5: The $\mathcal{F}_{\text{PT}}$ ideal functionality

Figure 2 shows the workings of POLAR in detail. The core idea of the protocol is making the clients use $\mathcal{F}_{\text{AB-MPC}}$ to compute the proximity testing function (Equation 1), but reducing their computation and communication overhead by letting the servers precompute the correlated randomess for them (using $\mathcal{F}_{\text{Out-MPC}}$). The first steps 1 and 2 provide the clients with the precomputed data, in the following steps the clients use $\mathcal{F}_{\text{AB-MPC}}$ to compute their desired functionality. The precomputed data is not used by the clients in the hybrid model with ideal functionalities available, but it is used when we replace the ideal functionalities by their implementations for the real-world implementation. In that case, the clients will use the precomputed data to speed-up their implementation of $\mathcal{F}_{\text{AB-MPC}}$.

Parameters: a positive number R, the radius of proximity testing.
Setup: `Alice`, `Bob` and the two servers. Alice and Bob access to the $\mathcal{F}_{\text{AB-MPC}}$ functionality, all four parties have access to $\mathcal{F}_{\text{Out-MPC}}$ functionality. `Alice` and `Bob` receive (x_a, y_a) and (x_b, y_b) as inputs.

1) The servers send `Eval` to $\mathcal{F}_{\text{Out-MPC}}$ functionality.
2) $\mathcal{F}_{\text{Out-MPC}}$ sends the precomputed data to the clients `Alice` and `Bob`.
3) The clients input their inputs x_a, y_a, x_b, y_b into the $\mathcal{F}_{\text{AB-MPC}}$.
4) The clients compute
$$[\![D]\!]_p \leftarrow ([\![x_a]\!]_p - [\![x_b]\!]_p)^2 + ([\![y_a]\!]_p - [\![y_b]\!]_p)^2$$
using `LinComb` and `Mult` operations of $\mathcal{F}_{\text{AB-MPC}}$.
5) The clients compare $[\![D]\!]_p$ to R^2 via
$$[\![\rho]\!]_2 \leftarrow \texttt{Compare}([\![D]\!]_p, R^2).$$
6) The clients output the $[\![\rho]\!]_2$ to `Alice`.

Figure 6: The POLAR protocol

4. Security Analysis

To prove the security of POLAR (Figure 6) we show that it securely implements the functionality $\mathcal{F}_{\text{PT}}$ (Figure 5) in the presence of static active adversary who can corrupt any subset of parties as long as one of the servers is not corrupted. Formally, it is stated by Theorem 1.

Theorem 1. The protocol POLAR *securely computes $\mathcal{F}_{PT}$ with abort in the presence of static malicious adversary* [27] who is allowed to corrupt any subset of parties as long as at least one of the servers is not corrupted.

Informally, the theorem above states that whatever an adversary (non-uniform polynomial time algorithm) can achieve by corrupting parties in POLAR, can be also achieved by some simulator who corrupted the same parties in $\mathcal{F}_{\text{PT}}$ (as long as the adversary does not capture both servers at the same time). In the case of trivial adversary which does not interfere with the protocol's execution, this ensures that both POLAR and $\mathcal{F}_{\text{PT}}$ produce the same result. This way, the $\mathcal{F}_{\text{PT}}$ serves as a specification of both correctness and security of POLAR; and whatever leakage is allowed by $\mathcal{F}_{\text{PT}}$, can also happen in POLAR. More details on this simulation paradigm can be found in the tutorial by Lindell [27].

Since the protocol POLAR is modular and is built from off-the-shelf MPC techniques (shown on Figure 3), our proof argument simply combines the proofs of the corresponding techniques. Showing here all the details would be too cumbersome and technical, therefore we only give an overview of the major steps (but we encourage a curious reader to go through the formal definitions of the used techniques [9], [11], [12], [21] and check the details). The main objective of this section is to show that the techniques from Figure 3 fit each other.

Combining of edaBits with Tinier and SPDZ is trivial, since the latter two are the standard MPC protocols working over their corresponding domains, and edaBits was intended to work with exactly this type of protocols. It is also worth noting that the combination of these three techniques is implemented out of the box by the MP-SPDZ [24] framework.

Combination of edaBits with the outsourcing of computation technique is not as straightforward: outsourcing of computation can be applied to either an arithmetic MPC or a binary one, but edaBits (which implements the ideal functionality $\mathcal{F}_{\text{AB-MPC}}$ shown on Figure 1) combines both. But it is easy to resolve since outsourcing can be applied to both binary and arithmetic domains independently, then both types of values can be outsourced.

We also claim that if the two servers are corrupted while both clients are honest, then only the correctness of the end result can be violated, but the adversary can learn nothing about the client inputs. This is due to servers never receiving any messages that would depend on client data; in fact, the protocol flow can be reordered so that all the interaction of clients with servers happens without clients ever using their inputs. Note that this claim is not captured by the statement of the Theorem 1, which requires one of the servers to stay uncorrupted.

5. Evaluation

To evaluate the performance of POLAR, we implemented the algorithms of servers and clients in the MP-SPDZ [24] cryptographic framework and made it available online [1]. We compare it to the performance of $\text{ABY}^{\text{C}}_{\text{AY}}$ and $\text{ABY}^{\text{C}}_{\text{Y}}$ [23], OLIC [31]. The former two are the state of the art in server-less proximity testing and the latter one is a server-aided protocol, but the servers in OLIC are involved in the protocol to a greater extent and actually do computations on the client data.

For the performance comparison, we focus on total execution time (on a single CPU core) and on total data exchanged by parties.

To achieve a more fair comparison, we ran all the protocols on the same Linux machine having Intel(R) Core(TM) i7-8700 CPU and 32 GB of RAM. We used the implementation provided by the original paper for each of the protocols: the C++ implementation using ABY [10] framework for $\text{ABY}^{\text{C}}_{\text{AY}}$ and $\text{ABY}^{\text{C}}_{\text{Y}}$, the Python implementation using the GMP library for OLIC. Although the protocols are implemented using different tools, the bulk of their computations is done by low-level C libraries (and the communication cost is independent of the tools), therefore such comparison is useful nevertheless. We do not introduce any intentional network latency. For each protocols, all the parties are executed on the same machine (one CPU core per party) and communicate through loopback network device. The following list shows the parameters with which we instantiated each of the protocols.

$\text{ABY}^{\text{C}}_{\text{AY}}$ and $\text{ABY}^{\text{C}}_{\text{Y}}$. We use ABY [10] parameters of the original paper [23]: `bits` $= 64$, `secparam` $= 128$. In other words, the values domain is 2^{64} and the symmetric security key length is 128 bits, as used by the original implementation. We do not increase the key lenght to keep our analysis conservative, since doing so could make these protocols slower.

OLIC. We use the most efficient one of the two instantiations presented in the original paper [31], namely, the **(EC)** which is based on Curve25519 and M383 elliptic curves.

POLAR. For the $\mathcal{F}_{\text{AB-MPC}}$ executed by the clients, we instantiate SPDZ and Tinier with the security parameter of 48 bits, and plaintext of values SPDZ consist of 64 bits.

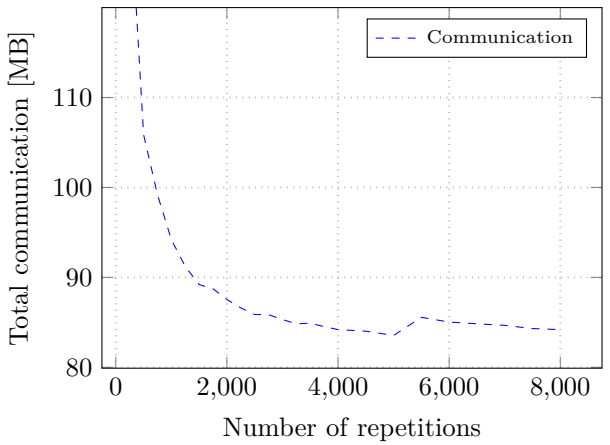
For the $\mathcal{F}_{\text{AB-MPC}}$ that is executed by the servers (as part of $\mathcal{F}_{\text{Out-MPC}}$), the SPDZ plaintext values are 64 bits (modulo a prime), and all the other parameters are as above. The statistical security parameter for edaBits is 40, the bucket size is $B = 4$. The preprocessing protocols used for SPDZ is MASCOT [25].

We do not include the performance of clients in our benchmarks of POLAR since it is negligible: the CPU time is under 2 ms while the total communication is under 2 KB.

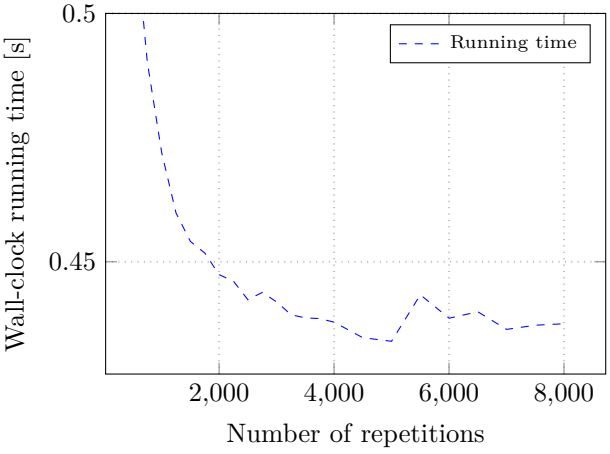
1. https://www.cse.chalmers.se/research/group/security/polar/

Figure 7 shows the amortized performance of servers in POLAR depending on the number of times the protocol is repeated. These measurements include both setup time and the actual protocol execution. As the number of repetitions approaches 5000, the amortized execution time reaches 0.4 seconds, and the total communication cost reaches 80.5 MB. We use these two numbers as constants in the next plots, where we compare POLAR to other protocols. (There is an unusual spike at 5200 repetitions, which we expect was caused by some internal details of MP-SPDZ library which we use. We speculate it could be due to MP-SPDZ generating precomputation data in large batches, and 5200 could be the threshold which causes an extra batch to be generated.)

The performance of OLIC depends on the specific value used for the radius R, this is reflected in the measurements presented on Figure 8. The protocols that have performance independent of R are shown there as straight horizontal lines. Notably, POLAR is less efficient than $\text{ABY}^{\text{C}}_{\text{AY}}$ and $\text{ABY}^{\text{C}}_{\text{Y}}$, but it still becomes more efficient than OLIC for large enough values of R. We consider it a minor price to pay given that POLAR is the only protocol that achieves malicious security (all other works are only passively



(a) Total communication



(b) Running time

Figure 7: Amortized performance of servers in POLAR by the number of repetitions

secure, and in OLIC the servers work with client data directly).

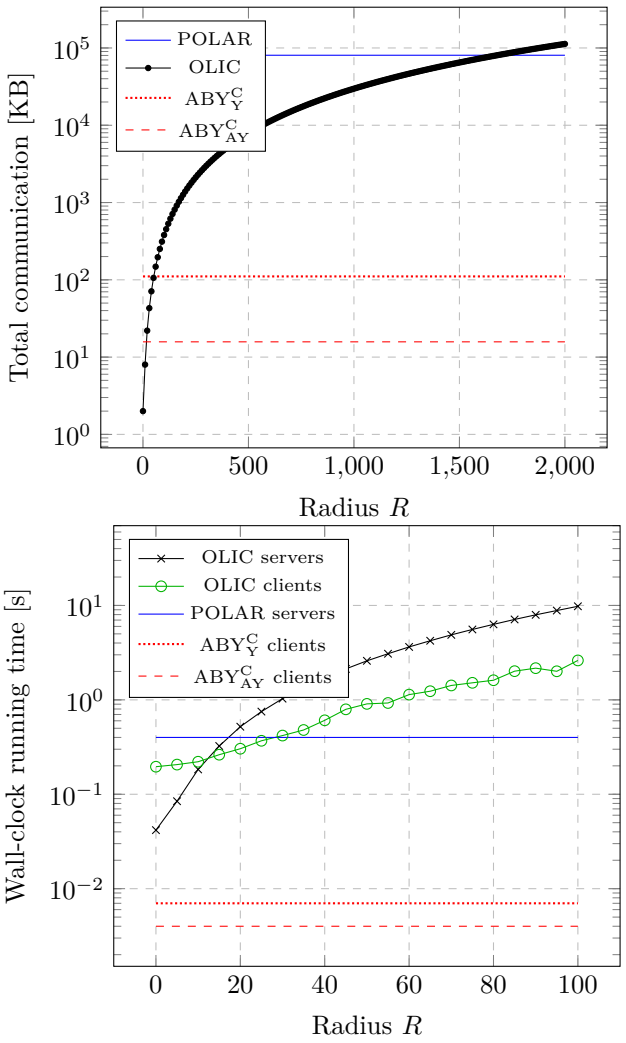


Figure 8: Comparison of POLAR with OLIC, ABY$^{C}_{Y}$ and ABY$^{C}_{AY}$

6. Related Work

Zhong et al. [41] propose the Louis, Lester and Pierre protocols for location proximity. The Louis protocol computes the distance between Alice and Bob using additively homomorphic encryption. It relies on a third party to perform the proximity test, and the thirst party is trusted with handling user data; in particular it learns the result of protocol execution. The Lester protocol does not use a third party but rather than performing proximity testing computes the actual distance between Alice and Bob. The Pierre protocol divides the space into a grid of cells and reveals the cell distance between Alice and Bob. All three protocols are only passively secure.

Narayanan et al. [29] present protocols for proximity testing. They cast the proximity testing problem as equality testing on a grid system of hexagons. One of the proposed protocols utilizes an oblivious server that aids clients in computations on their data. Parties in this protocol uses symmetric encryption, which leads to better performance. However, this requires having preshared keys among parties, which is less amenable to one-to-many proximity testing. Saldamli et al. [36] build on the protocol with the oblivious server and suggest optimizations based on properties from geometry and linear algebra. Nielsen et al. [30] and Kotzanikolaou et al. [26] also propose grid-based solutions.

Hide&Crypt by Freni et al. [13] splits proximity into two steps. First, it performs filtering between a third party and the initiating principal. Second, the two principals execute computation to achieve finer granularity. In both steps, the granule in which a principal is located is sent to the other party. C-Hide&Hash by Mascetti et al. [28] is a centralized protocol, where the principals do not need to communicate pairwise but otherwise share many aspects with Hide&Crypt. FriendLocator by Šikšnys et al. [39] is a centralized protocol where clients map their positions to different granularities, similarly to Hide&Crypt, but instead of refining via the second principal, each iteration is done via the third party. VicinityLocator also by Šikšnys et al. [38] is an extension of FriendLocator, which allows the proximity of a principal to be represented not only in terms of any shape.

Šeděnka and Gasti [37] homomorphically compute distances using the UTM projection, ECEF (Earth-Centered Earth-Fixed) coordinates, and the Haversine formula that makes it possible to consider the curvature of the Earth. Hallgren et al. [18] introduce InnerCircle for parallelizable decentralized proximity testing. They use linerly homomorphic encryption to perform proximity testing between two clients. The MaxPace [19] protocol builds on the speed constraints of an InnerCircle-style protocol as to limit the effects of trilateration attacks. Polakis [33] study different distance and proximity disclosure strategies employed in the wild and experiment with practical effects of trilateration.

Sakib and Huang [35] explore proximity testing using elliptic curves. They do not use any third party. Järvinen et al. [23] design efficient schemes for Euclidean distance-based privacy-preserving location proximity, as well as schemes for polygon-based matching. They demonstrate performance improvements over InnerCircle. Yet their protocol offers only passive security. Hallgren et al. [17] show how to leverage proximity testing for endpoint-based ridesharing, building on the InnerCircle protocol (and also being only passively secure), and compare this method with a method of matching trajectories. Oleynikov et al. [31] build OLIC, a natural extension of InnerCircle to the two-server setting to perform Euclidean distance based matching. They also propose the "napping party" model with two servers that formalizes the possibility for parties to submit their locations at independent moments of time. The "napping party" setting requires that the clients communicate with servers at disjoint intervals of time and that they do not share any secret data (e.g. cryptographic keys) before the protocol starts. It is necessary to have at

least two servers to achieve this property. As shown by Hallevi et al. [16], using one server for this purpose will leak the clients' data to it. Further works on generic MPC in client-server settings [1], [2], [14], [15], [22] also consider one-server scenarios. Even though in OLIC the clients use the help of the two servers in order to run the protocol, the computational and communicational requrements on clients there are quite high. And the servers still handle (encrypted, masked) clients data; and in case both server collude, they will be able to learn some extra information about client data.

The main challenge of Euclidean distance based proximity testing is efficiently combining the arithmetic operations (like computing the squared distance) with the comparison operation; many existing tools for multiparty computation tend to be efficient only for one of the two kinds of operations, and performing the other one introduces great overhead. We overcome this in our POLAR by mixing the MASCOT [25] and Tinier [12] MPC protocols for computations in the two different domains, and the edaBits [11] technique to convert between the two domains. All three mentioned techniques are quite efficient in the amortized sense: i.e. when the number of MPC operations done is high, the cost of running the MPC protocol per operation is low. This is a major challenge to applying these techniques to a client-client setting where clients want to compute a relatively simple functionality. We overcome this obstacle by introducing a pair of servers and letting the servers do the heavy precomputation phase for many different pairs of clients in one batch, taking advantage of the amortization.

Very recently, in lieu of preventing the spreading of COVID-19, privacy-preserving proximity testing witness a boom of protocols that rely on Bluetooth communication, e.g., [5], [40]. These solutions realize proximity testing without relying on knowing the exact location of clients. Such solutions are effective only for shorter radius (Bluetooth range) and the distance between users cannot be accurately computed (e.g., signal strength varies in the presence of physical barriers and with weather conditions). In contrast, this work does not want to rely on a specific technology (e.g., Bluetooth communication) and aims at providing precise matching using the Euclidean distance. We remark that purely protocol-based solutions which are the focus on this work aim to privately implement the partial functionality of global services like social networks, messengers and taxi services.

To summarize, most [13], [18], [19], [23], [29], [31], [35], [36], [37], [38], [39], [41] of the existing approaches to proximity testings offer protocols with limited practical applicability since they either not actively secure, or are too heavy to be executed to resource-constrained clients. POLAR is also modular and is composed of state of the art MPC techniques, so any performance improvement to those techniques will automatically improve POLAR as well.

7. Discussion

Assumptions. POLAR is not yet a fully-featured protocol to implement LBS out of the box. Rather, it is best seen as a fundamental building block that can be used by a privacy-enhancing LBS. It works in the standard setting of MPC protocols [27], the same setting was used for a number of previous proximity testing protocols [18], [23], [31], [32] albeit the (passive) adversary was more limited in those protocols. The assumptions of this model are: parties communicate through secure point-to-point channels (can be implemented in real life by means of Public Key Infrastructure), in the beginning of the protocol the (active) adversary can corrupt some of the parties and arbitrarily change their behavior attempting to learn something about the other parties' inputs and cause the other parties' outputs to be incorrect. As long as one of the servers is honest, POLAR ensures the security and the correctness of the protocol. But even if both servers are corrupted, they can only interfere with the protocol result, but not learn anything about client data. The only case when the adversary can infer something about client inputs is if both servers and one of the clients is corrupted at the same time.

Scope. The setting of POLAR does not address the data leakage inherit to the functionality itself, e.g., knowing whether some user is close to you or not inevitably reveals something about that user's location, or when two users perform the matching the servers will learn the fact that matching happened (since they know what users they communicated with and when) but not the result of that matching.

Generalizations. Our approach is trivially augmentable to support time-based matching [32], i.e. to allow clients to submit the time interval during which they plan to be in the specified location and make the protocol match them only if the locations are close *and* the time intervals intersect. This can be useful for friend-finding services as well as ridesharing and taxi applications (e.g. BlaBlaCar [3]), where drivers need to be close to pick up the passengers at the right time (and get the actual passenger location if the matching succeeded).

POLAR can be easily generalized to use more than two servers, so that it stays secure as long as at least one of the servers is honest. This significantly weakens the security assumption it depends on, making the protocol more reliable at a cost of certain server-side performance overhead. Since the real-life purpose of having two servers was to allow distributing trust between two independent organizations that are providing the LBS together, distributing it over a larger number of organizations makes the task of compromising the whole system a lot harder.

8. Conclusion

We presented POLAR, a secure and privacy-enhancing protocol for proximity testing, which performs exact Euclidean distance based matching. POLAR introduces two servers to the client-to-client setting to aid in protocol. This allows the clients to run

an actively secure MPC protocol offloading the MPC protocol's precomputations to the servers.

Our evaluation results confirm that the amortized performance of POLAR is practical: the clients running time per client pair is close to negligible, and the communication cost is around 84 MB (if amortized over 2000 repetitions) which is acceptable given that in real life the servers will not be as resource constrained as clients; and 84 MB is equivalent to streaming a short video.

We leave a more extensive evaluation of POLAR's performance in the presence of realistic network latency for the future work, as well as the evaluation of time-based matching. Another direction for future work is to apply the proposed server-aided precomputation technique to other problems where resource-constrained clients want to run a lightweight MPC protocol.

Acknowledgments This work was partially supported by the Wallenberg AI, Autonomous Systems and Software Program (WASP) funded by the Knut and Alice Wallenberg Foundation, the Swedish Foundation for Strategic Research (SSF), the Swedish Research Council (VR), and the Excellence Center at Linköping – Lund in Information Technology (ELLIIT).

References

[1] Amos Beimel, Ariel Gabizon, Yuval Ishai, Eyal Kushilevitz, Sigurd Meldgaard, and Anat Paskin-Cherniavsky. Non-interactive secure multiparty computation. In Juan A. Garay and Rosario Gennaro, editors, *CRYPTO*, volume 8617 of *LNCS*, pages 387–404. Springer, 2014.

[2] Fabrice Benhamouda, Hugo Krawczyk, and Tal Rabin. Robust non-interactive multiparty computation against constant-size collusion. In Jonathan Katz and Hovav Shacham, editors, *CRYPTO*, volume 10401 of *LNCS*, pages 391–419. Springer, 2017.

[3] BlaBlaCar - Trusted carpooling. https://www.blablacar.com/.

[4] Ran Canetti. Security and composition of multi-party cryptographic protocols. Cryptology ePrint Archive, Report 1998/018, 1998. https://eprint.iacr.org/1998/018.

[5] Claude Castelluccia, Nataliia Bielova, Antoine Boutet, Mathieu Cunche, Cédric Lauradoux, Daniel Le Métayer, and Vincent Roca. Robert: Robust and privacy-preserving proximity tracing. 2020.

[6] Samantha Cole. Yahoo engineer used insider access to get private photos of women. https://www.vice.com/en/article/59nwyk/yahoo-engineer-used-insider-access-to-get-private-photos-of-women, 2019. [Online; accessed 16-May-2021].

[7] Joseph Cox. Snapchat employees abused data access to spy on users. https://www.vice.com/en/article/xwnva7/snapchat-employees-abused-data-access-spy-on-users-snaplion, 2019. [Online; accessed 16-May-2021].

[8] Joseph Cox and Max Hoppenstedt. Sources: Facebook has fired multiple employees for snooping on users. https://www.vice.com/en/article/bjp9zv/facebook-employees-look-at-user-data, 2018. [Online; accessed 16-May-2021].

[9] Ivan Damgård, Valerio Pastro, Nigel Smart, and Sarah Zakarias. Multiparty computation from somewhat homomorphic encryption. In Reihaneh Safavi-Naini and Ran Canetti, editors, *Advances in Cryptology – CRYPTO 2012*, pages 643–662, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[10] Daniel Demmler, Thomas Schneider, and Michael Zohner. ABY - A framework for efficient mixed-protocol secure two-party computation. In *22nd Annual Network and Distributed System Security Symposium, NDSS 2015, San Diego, California, USA, February 8-11, 2015*. The Internet Society, 2015.

[11] Daniel Escudero, Satrajit Ghosh, Marcel Keller, Rahul Rachuri, and Peter Scholl. Improved primitives for mpc over mixed arithmetic-binary circuits. Cryptology ePrint Archive, Report 2020/338, 2020. https://eprint.iacr.org/2020/338.

[12] Tore Kasper Frederiksen, Marcel Keller, Emmanuela Orsini, and Peter Scholl. A unified approach to mpc with preprocessing using ot. Cryptology ePrint Archive, Report 2015/901, 2015. https://eprint.iacr.org/2015/901.

[13] Dario Freni, Carmen Ruiz Vicente, Sergio Mascetti, Claudio Bettini, and Christian S. Jensen. Preserving location and absence privacy in geo-social networks. In *CIKM*, pages 309–318, 2010.

[14] S. Dov Gordon, Tal Malkin, Mike Rosulek, and Hoeteck Wee. Multi-party computation of polynomials and branching programs without simultaneous interaction. In Thomas Johansson and Phong Q. Nguyen, editors, *EUROCRYPT*, LNCS. Springer, 2013.

[15] Shai Halevi, Yuval Ishai, Abhishek Jain, Ilan Komargodski, Amit Sahai, and Eylon Yogev. Non-interactive multiparty computation without correlated randomness. In Tsuyoshi Takagi and Thomas Peyrin, editors, *ASIACRYPT*, volume 10626 of *LNCS*, pages 181–211. Springer, 2017.

[16] Shai Halevi, Yehuda Lindell, and Benny Pinkas. Secure computation on the web: Computing without simultaneous interaction. In *CRYPTO*, pages 132–150, 2011.

[17] P. Hallgren, C. Orlandi, and A. Sabelfeld. PrivatePool: Privacy-Preserving Ridesharing. In *CSF*, pages 276–291, Aug 2017.

[18] Per A. Hallgren, Martín Ochoa, and Andrei Sabelfeld. InnerCircle: A parallelizable decentralized privacy-preserving location proximity protocol. In *PST*, pages 1–6, 2015.

[19] Per A. Hallgren, Martín Ochoa, and Andrei Sabelfeld. MaxPace: Speed-Constrained Location Queries. In *CNS*, 2016.

[20] Alex Hern. Uber employees 'spied on ex-partners, politicians and beyoncé'. https://www.theguardian.com/technology/2016/dec/13/uber-employees-spying-ex-partners-politicians-beyonce, 2016. [Online; accessed 16-May-2021].

[21] Thomas P Jakobsen, Jesper Buus Nielsen, and Claudio Orlandi. A framework for outsourcing of secure computation. In *Proceedings of the 6th edition of the ACM Workshop on Cloud Computing Security*, pages 81–92, 2014.

[22] Ayman Jarrous and Benny Pinkas. Canon-mpc, a system for casual non-interactive secure multi-party computation using native client. In Ahmad-Reza Sadeghi and Sara Foresti, editors, *WPES*, pages 155–166. ACM, 2013.

[23] Kimmo Järvinen, Agnes Kiss, Thomas Schneider, Oleksandr Tkachenko, and Zheng Yang. Faster privacy-preserving location proximity schemes for circles and polygons. *IET Information Security*, 14, 10 2019.

[24] Marcel Keller. Mp-spdz: A versatile framework for multi-party computation. Cryptology ePrint Archive, Report 2020/521, 2020. https://eprint.iacr.org/2020/521.

[25] Marcel Keller, Emmanuela Orsini, and Peter Scholl. Mascot: Faster malicious arithmetic secure computation with oblivious transfer. In *Proceedings of the 2016 ACM SIGSAC Conference on Computer and Communications Security*, CCS '16, page 830–842, New York, NY, USA, 2016. Association for Computing Machinery.

[26] Panayiotis Kotzanikolaou, Constantinos Patsakis, Emmanouil Magkos, and Michalis Korakakis. Lightweight private proximity testing for geospatial social networks. *Computer Communications*, 73:263–270, 2016.

[27] Yehuda Lindell. How to simulate it - a tutorial on the simulation proof technique. Cryptology ePrint Archive, Report 2016/046, 2016. https://eprint.iacr.org/2016/046.

[28] Sergio Mascetti, Dario Freni, Claudio Bettini, Xiaoyang Sean Wang, and Sushil Jajodia. Privacy in geo-social networks: proximity notification with untrusted service providers and curious buddies. *VLDB J.*, 20(4):541–566, 2011.

[29] Arvind Narayanan, Narendran Thiagarajan, Mugdha Lakhani, Michael Hamburg, and Dan Boneh. Location privacy via private proximity testing. In *NDSS*, 2011.

[30] Janus Dam Nielsen, Jakob Illeborg Pagter, and Michael Bladt Stausholm. Location privacy via actively secure private proximity testing. In *PerCom Workshops*, pages 381–386. IEEE CS, 2012.

[31] Ivan Oleynikov, Elena Pagnin, and Andrei Sabelfeld. Where are you Bob? Privacy-Preserving Proximity Testing with a Napping Party. In *ESORICS*, 2020.

[32] Elena Pagnin, Gunnar Gunnarsson, Pedram Talebi, Claudio Orlandi, and Andrei Sabelfeld. TOPPool: Time-aware Optimized Privacy-Preserving Ridesharing. *PoPETs*, 2019(4):93–111, 2019.

[33] Iasonas Polakis, George Argyros, Theofilos Petsios, Suphannee Sivakorn, and Angelos D. Keromytis. Where's wally?: Precise user discovery attacks in location proximity services. In *CCS*, 2015.

[34] Dragos Rotaru and Tim Wood. Marbled circuits: Mixing arithmetic and boolean circuits with active security. Cryptology ePrint Archive, Report 2019/207, 2019. https://eprint.iacr.org/2019/207.

[35] Muhammad N. Sakib and Chin-Tser Huang. Privacy preserving proximity testing using elliptic curves. In *ITNAC*, pages 121–126. IEEE Computer Society, 2016.

[36] Gökay Saldamli, Richard Chow, Hongxia Jin, and Bart P. Knijnenburg. Private proximity testing with an untrusted server. In *WISEC*, pages 113–118. ACM, 2013.

[37] Jaroslav Sedenka and Paolo Gasti. Privacy-preserving distance computation and proximity testing on earth, done right. In *AsiaCCS*, pages 99–110, 2014.

[38] Laurynas Siksnys, Jeppe Rishede Thomsen, Simonas Saltenis, and Man Lung Yiu. Private and flexible proximity detection in mobile social networks. In *MDM*, pages 75–84, 2010.

[39] Laurynas Siksnys, Jeppe Rishede Thomsen, Simonas Saltenis, Man Lung Yiu, and Ove Andersen. A location privacy aware friend locator. In *SSTD*, pages 405–410, 2009.

[40] Carmela Troncoso, Mathias Payer, Jean-Pierre Hubaux, Marcel Salathé, James Larus, Edouard Bugnion, Wouter Lueks, Theresa Stadler, Apostolos Pyrgelis, Daniele Antonioli, et al. Decentralized privacy-preserving proximity tracing. *arXiv preprint arXiv:2005.12273*, 2020.

[41] Ge Zhong, Ian Goldberg, and Urs Hengartner. Louis, lester and pierre: Three protocols for location privacy. In *PET*, pages 62–76, 2007.