Thesis for The Degree of Licentiate of Engineering

Server-Aided Privacy-Preserving Proximity Testing

Ivan Oleynikov

CHALMERS

Division of Computing Science, Information Security
Department of Computer Science & Engineering
Chalmers University of Technology
Gothenburg, Sweden, 2022

Server-Aided Privacy-Preserving Proximity Testing

Ivan Oleynikov

Department of Computer Science & Engineering
Division of Computing Science, Information Security
Chalmers University of Technology
Gothenburg, Sweden

This thesis has been prepared using LaTeX.
Printed by Chalmers Reproservice,
Gothenburg, Sweden 2022.

Abstract

Proximity testing is at the core of many Location-Based online Services (LBS) which we use in our daily lives to order taxis, find places of interest nearby, connect with people. Currently, most such services expect a user to submit his location to them and trust the LBS not to abuse this information, and use it only to provide the service. Existing cases of such information being misused (e.g., by the LBS employees or criminals who breached its security) motivates the search for better solutions that would ensure the privacy of user data, and give users control of how their data is being used.

In this thesis, we address this problem using cryptographic techniques. We propose three cryptographic protocols that allow two users to perform proximity testing (check if they are close enough to each other) with the help of two servers.

In the papers 1 and 2, the servers are introduced in order to allow users not to be online at the same time: one user may submit their location to the servers and go offline, the other user coming online later and finishing proximity testing. The drastically improves the practicality of such protocols, since the mobile devices that users usually run may not always be online. We stress that the servers in these protocols merely aid the users in performing the proximity testing, and none of the servers can independently extract the user data.

In the paper 3, we use the servers to offload the users' computation and communication to. The servers here pre-generate correlated random data and send it to users, who can use it to perform a secure proximity testing protocol faster. Paper 3, together with the paper 2, are highly practical: they provide strong security guarantees and are suitable to be executed on

resource-constrained mobile devices. In fact, the work of clients in these protocols is close to negligible as most of the work is done by servers.

Keywords: privacy, secure proximity testing, MPC, active security

Acknowledgments

First of all, I thank my supervisors Andrei and Elena for all the guidance and support during these three years I spent at Chalmers. I am very grateful for your patience, help with organizing myself I got from you and all the valuable feedback. Working with you greatly helped me fight my procrastination and the sense of helplessness in the face of challenging problems. I appreciate all the effort you made for me.

Big thank you to all my friends from Chalmers for their hospitality and help with settling down, getting me out of my introverted moods, showing new hobbies, challenging my views, and making Gothenburg feel like home. Mohammad, Iulia, Jeremy, Benjamin, Alexander, Carlos, Nachi, Max, Fabian, Matti, Irene, Elizabet, Augustin, Liam, Boel, Carlo, Jeff and many more from CS division.

I owe any knowledge and taste in Computer Science that I have to my teachers and classmates from Academic University in St. Petersburg, Russia. I am greatly privileged to have learned from you, and without those two difficult years I would never end up doing Computer Science.

I am very greatful to my buddies and instructors from Brazilian Jiu-Jitsu at Fighter Centre for giving me sense of community and showing example of dedication and hard work.

Большое спасибо моей семье за помощь и поддержку, без которых у меня бы ничего не получилось.

This page intentionally left blank.

Contents

Introduction

An increasing number of online services depend on the information about the location of their users, be it to deliver a taxi home, suggest restaurants in proximity to visit, build a route on the map, connect with people of interest closeby. In current practice, such Location-Based Services (LBS) are full-trust centralized services. They require the user (usually through a mobile device like smartphone) to submit her location to the LBS and trust that the service will not misuse or accidentally leak it to another party.

In principle, the functioning of many LBS does not depend on knowing the exact user locations, since it uses only the result of some function over them. For example, it would suffice for a dating service to know which users are close enough for a match with a given user even if the exact coordinates of its users are not given. This, together with existing precedents of LBS—maliciously or by accident—misusing user location data [6, 8, 10, 11, 16], motivates the search for better ways to implement LBS that would ensure the privacy of user locations.

The goal of this thesis is to address some of these privacy issues using cryptographic tools.

Cryptographic Protocols

The functionality of an LBS is built on the exchange of messages over internet between user's mobile device and the servers of LBS. Therefore it is natural to see this functionality as a communication protocol, and try to provide privacy and security of the parties involved in it using cryptography.

In cryptography, there are a few standard definitions of security for a cryptographic protocol, which vary in the strength of the adversary they

tolerate and versatility of settings where they can work. In the following, we give a high-level overview of the definitions that are relevant to this work, more details can be found in tutorial by Lindell [12].

Passive security. When the protocol starts, parties hold their inputs, and the goal of the protocol is to reach a state where they would hold their corresponding outputs which are defined as (probabilistic) functions of all parties' inputs. Formally, the functionality of a protocol (i.e. the computation that the protocol is expected to carry out) is defined by a random function

$$f(i_1, i_2, \dots i_n) = (o_1, o_2 \dots o_n).$$

Here, i_j is the input of the party i and o_j is its output.

Passive security requires that party j (who honestly follows all the protocol steps, but tries to passively extract as much information as it can from the messages it receives) should not be able to deduce anything about the inputs of the other parties other than what is already leaked by its corresponding output o_j. Formally, this is proven by showing that all the messages that the party receives can be efficiently simulated given only the party's input and output values. The simulated messages are required to be computationally indistinguishable from the actual ones that are sent during the protocol execution, which guarantees that the actual messages do not carry more information than the simulted ones, which in turn do not carry more information than the input and output of the corresponding party.

This definition also covers the case when multiple parties collude in order to extract infromation about the inputs of the others.

Active security. This notion of security ensures that even if some parties deviate from the protocol specification, they will not learn anything about the inputs of the others nor they will cause the protocol to produce an incorret result.

Here, the security formalism is based on the model of Real and Ideal worlds. Real world setup corresponds to the actual executions that can happen in practice, it involves all the parties present in the protocol and they are allowed to exchange messages over network. By default, the parties here correctly execute the algorithms assigned to them by the protocol specification. When the execution completes, each party algorithm returns the output value intended for that party. The model also considers executions where the adversary corrupts one or more

of the parties and replaces their algorithms by algorithms of its choice, possibly making the parties (both corrupted and uncorrupted) to produce a different output at the end of protocol execution. Ideal world setup involves all the parties and also a special extra party called Functionality, which can never be corrupted and is completely trusted by everyone.

Ideal world executions are not intended to be implemented in practice, but they serve as a specification of how we want the Real world to behave. When no parties are corrupted, we require that for any tuple of inputs $(i_1, i_2 \dots i_n)$ the outputs of the protocol in the Ideal world are indistinguishable from the corresponding outputs in the Real world, denoted as

$$\text{Ideal}(i_1, i_2 \dots i_n) \approx \text{Real}(i_1, i_2 \dots i_n).$$

The $\approx$ here denotes computational indistinguishability.

For the case when an adversary A corrupts some parties in the Ideal world, there must exist a simulator S that corrupts the same parties in the Real world, and causes both worlds to produce the same output,

$$\text{Ideal}^S(i_1, i_2 \dots i_n) \approx \text{Real}^A(i_1, i_2 \dots i_n).$$

The Ideal world specification (which includes the algorithms for Functionality and all the parties) serves as a definition of what we want the protocol to compute, while the Real world specification (the algorithms of all the parties) is the solution showing how to achieve that in practice.

Universally Composable security. Universally Composable security adds an extra party called Environment to both Ideal and Real worlds of Active security. Now the adversary and simulator are allowed to interact with it, and the security requirement demands that no Environment should be able to tell whether it is connected to Ideal or Real world.

In other words, Universally Composable security is a modification of Active security that is indistinguishable by an interactive distinguisher. This simple generalization has an important practical effect, now the protocol is guaranteed to stay secure even if the parties are interacting in some way outside of it, for example, running another instance of the same protocol in parallel, or composing multiple protocols to build a bigger one. The intuition here is that the Environment represents any interaction that the parties (including the adversary) might have outside of the current protocol instance.

> Protocols that are proven to be Universally Composable secure can be composed together with little effort, yielding another Universally Composable protocol.

In this thesis we construct protocols that achieve Active and Passive security, while some of the prior work we use in our construction meets the definition of Universally Composable security.

Proximity Testing

A great number of LBS, at their core, use the location data to test if some locations are close enough to each other. For example, a taxi service would take a passenger's location and look for a driver that is close enough (or the closest) to pick up the passenger. A ridesharing service would do the same, but consider the proximity of both start and end points of the journey (because here, the driver wants to take only those passengers whose trips are close to his).

Proximity Testing (PT) is range of problems of constructing secure cryptographic protocols that would allow two parties who know each their own location, to test if they are close enough to one another. PT is a generic term which covers different notions of what "location" is, what metric is used by "close enough" and what is a "secure" protocol. For example, PT for a taxi service could define location as a node in the graph representing the city road network, with the closeness metric being the shortest distance the driver has to travel to reach the passenger. An IoT system that wants to know if the user is home to turn on the lights, could represent the user location as a pair of GPS coordinates and test it for closeness to home via Euclidean distance.

In this work, we focus only on PT based on Euclidean distance. Here, the location of a user is a point on a plane represented by a pair of integer coordinates $p_1 = (x, y)$ and the PT protocol wants to test whether the distance between two such points p_1 and p_2 exceeds the pre-defined threshold value R or not.

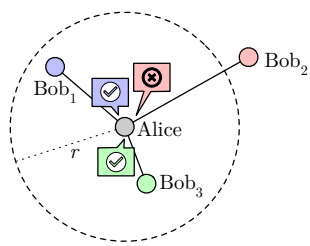


Figure 1: Illustration of Euclidean distance based matching

Euclidean distance based PT is relatively simple, it can be expressed via a few arithmetic operations and a comparison. This notion of PT arises naturally in some LBS (for example, dating apps or IoT system discussed above). And it can serve as a reasonable approximation of some other metrics (for example, given how much simpler Euclidean distance is

comapred to graph distance, a taxi service may want to approximate road distance with straight-line Euclidean distance). Euclidean distance based PT can also be used to compute closeness of other data represented by vectors of numbers, for example preferences of two users in a social network.

Generic Multi-Party Computation (MPC)

Generic Multi-Party Computation is a subfield of cryptography that aims at computing generic functions (often described by airthmetic, Boolean or mixed circuits) in the setting of cryptographic protocols, meeting the security properties like the ones we outlined above.

Since Euclidean distance based matching wants to compute a specific function privately, MPC can be seen as its generalization (computing arbitrary function) and a natural tool to apply on PT.

Thesis structure

In this thesis we construct three protocols for Euclidean distance based PT. Each of the three protocols allows two clients to perform the Euclidean distance based proximity testing with the help of two servers. In the first two papers, the servers are introduced to allow matching clients who are not online at the same time. In the third paper, the role of servers is to offload some part of computational and communicational overhead from the clients, but do so without sacrificing clients' privacy. High-level relation of the presented protocols is shown on Figure 2.

We build on two previously existing works on Euclidean distance based proximity testing, InnerCircle by Hallgren et al. [5] and ABY-based protocols of Järvinen et al. [9].

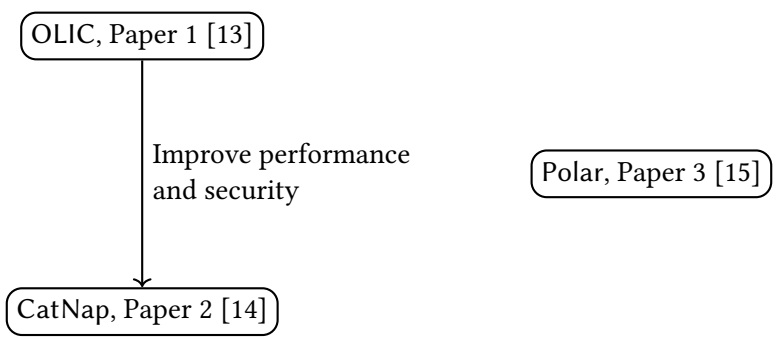


Figure 2: The relations of the protocols built in this thesis to each other

Paper 1: Where are you Bob? Privacy-Preserving Proximity Testing with a Napping Party [13]

In this paper, we construct OLIC, a protocol for privacy-preserving PT. This protocol involves two clients, Alice and Bob, who know their corrdinates (x_a, y_a) and (x_b, y_b) respectively, and they want Alice to know if the squared distance between them $D = (x_a - x_b)^2 + (y_a - y_b)^2$ exceeds R^2. The protocol has practical performance, but only for small values of radius R. Its running time and total communication grow proportionally to R^2. OLIC achieves Passive security.

The distionguishing feature of this protocol is allowing clients to perform the protocol without necessarily being online at the same time—we call it "offline" feature. To achieve that, we introduce two servers into the setting, and make Bob submit his location to the servers (in a privacy-preserving way, so none of the servers can read it), and have Alice come online later and finish the protocol with the servers without him being online.

The techniques used by OLIC are heavily based on the previously existing server-less protocol InnerCircle [5].

Statement of contributions We published this paper in collaboration with my supervisors Elena Pagnin and Andrei Sabelfeld. I implemented the protocol benchmarking code with some high-level guidance from Elena and Andrei, I also drafted the protocol description and its security proof, which we then heavily edited in a few iterations.

Appeared in: 25th European Symposium on Research in Computer Security (ESORICS)'2022.

Paper 2: CatNap: Leveraging Generic MPC for Actively Secure Privacy-Enhancing Proximity Testing with a Napping Party [14]

In this paper, we solve the same problem of Euclidean distance based PT with "offline" feature as in the previous paper, improving both security and performance compared to OLIC. The protocol CatNap that we build here has Active security and performance independent of radius R. We achieve this using the generic techniques from MPC, as opposed to ad-hoc techniques of OLIC.

The construction of CatNap is highly modular, it applies the used MPC techniques [1, 2, 3, 7] in a black-box way. This means that if some of those techniques get improved in the future, CatNap will get improved "for free". This work shows what can be achieved using off-the shelf techniques, and sets the bar for any future work based on ad-hoc techniques.

Statement of contributions I suggested the idea of trying what can be done for PT using MPC techniques, and implemented all the code and most of the security proof occasionally asking Elena and Andrei for a review and advice. I also prepared the draft of the paper, which we then discussed and edited together.

Appeared in: Proceedings of the 19th International Conference on Security and Cryptography'2022.

Paper 3: Outsourcing MPC Precomputation for Location Privacy [15]

This paper approaches the Euclidean distance based PT in the client-client setting (that does not rely on servers), using Actively secure MPC techniques [2, 3, 7, 9]. We argue that one of the main obstacles to applying the existing MPC protocols to client-client setting is performance. Modern implementations of MPC protocols are often tailored for good amortized performance when doing large amounts of computations, but become relatively expensive when applied to a simple functionality like Euclidean distance based matching.

In this paper, we seek a different compromise between performance and privacy and introduce two servers to speed up the PT that clients perform between themselves—albeit at a small privacy cost. Most MPC protocols rely on a resource-heavy precomputation phase that is performed without inputs from the parties. In this protocol, we offload the clients' precomputation phase onto the two servers which perform it inside another MPC protocol, and return the clients their precomputation data in privacy-preserving way (without servers themselves being able to learn it). This drastically speeds up the execution of clients' MPC protocol at a cost of introducing an extra assumption that that both servers will not get corrupted at the same time.

The burden of doing client precomputation inside an extra MPC protocol on the servers is quite moderate, since in this model the same pair of servers can serve multiple pairs of clients and generate the precomputation data for all of them in one large batch taking advantage of the MPC protocols' good amortized performance.

Statement of contributions I am the main author of this paper. I suggested the idea of offloading the precomputation for PT to servers which we refined in later discussions together, and also implemented all the code for this paper. I prepared the final draft of the paper, which we then proof-read and edited together.

Appeared in: IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)'2022, Location Privacy Workshop.

Bibliography

[1] R. Cramer, I. Damgård, D. Escudero, P. Scholl, and C. Xing. Spdz2k: Efficient mpc mod 2^k for dishonest majority. In *CRYPTO*, 2018.

[2] D. Escudero, S. Ghosh, M. Keller, R. Rachuri, and P. Scholl. Improved primitives for mpc over mixed arithmetic-binary circuits. In *CRYPTO*, 2020.

[3] T. K. Frederiksen, M. Keller, E. Orsini, and P. Scholl. A unified approach to mpc with preprocessing using ot. In *ASIACRYPT*, 2015.

[4] C. Gentry, S. Halevi, and V. Vaikuntanathan. i-hop homomorphic encryption and rerandomizable yao circuits. In *CRYPTO*. Springer, 2010.

[5] P. A. Hallgren, M. Ochoa, and A. Sabelfeld. InnerCircle: A parallelizable decentralized privacy-preserving location proximity protocol. In *PST*, 2015.

[6] A. Hern. Uber employees 'spied on ex-partners, politicians and Beyoncé', 2016. https://www.theguardian.com/technology/2016/dec/13/uber-employees-spying-ex-partners-politicians-beyonce.

[7] T. P. Jakobsen, J. B. Nielsen, and C. Orlandi. A framework for outsourcing of secure computation. In *ACM CCSW*, 2014.

[8] R. Jones. The Uber scammers who take users for a (very expensive) ride. 2016. https://www.theguardian.com/money/2016/apr/22/uber-scam-hacking-account-phantom-journeys.

[9] K. Järvinen, A. Kiss, T. Schneider, O. Tkachenko, and Z. Yang. Faster privacy-preserving location proximity schemes for circles and polygons. *IET Information Security*, 14, 10 2019.

[10] D. Lee. Uber concealed huge data breach, 2017. http://www.bbc.com/news/technology-42075306.

[11] S. Levin. Facebook fires engineer accused of stalking, possibly by abusing data access. 2018. https://www.theguardian.com/technology/2018/may/02/facebook-engineer-fired-alleged-stalker-tinder.

[12] Y. Lindell. How to simulate it–a tutorial on the simulation proof technique. In *Tutorials on the Foundations of Cryptography*, pages 277–346. Springer, 2017.

[13] I. Oleynikov, E. Pagnin, and A. Sabelfeld. Where are you Bob? Privacy-Preserving Proximity Testing with a Napping Party. In *Computer Security – ESORICS 2020*, pages 677–697, 2020.

[14] I. Oleynikov, E. Pagnin, and A. Sabelfeld. CatNap: Leveraging Generic MPC for Actively Secure Privacy-Enhancing Proximity Testing with a Napping Party. In *Proceedings of the 19th International Conference on Security and Cryptography – SECRYPT*, pages 237–248, 2022.

[15] I. Oleynikov, E. Pagnin, and A. Sabelfeld. Outsourcing MPC Precomputation for Location Privacy. In *2022 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*, pages 504–513, 2022.

[16] C. Shu. Uber reportedly tracked Lyft drivers using a secret software program named 'Hell'. https://techcrunch.com/2017/04/12/hell-o-uber/, 2017.

1

Where are you Bob? Privacy-Preserving Proximity Testing with a Napping Party

Ivan Oleynikov, Elena Pagnin, Andrei Sabelfeld

ESORICS'2020

Abstract. Location based services (LBS) extensively utilize proximity testing to help people discover nearby friends, devices, and services. Current practices rely on full trust to the service providers: users share their locations with the providers who perform proximity testing on behalf of the users. Unfortunately, location data has been often breached by LBS providers, raising privacy concerns over the current practices. To address these concerns previous research has suggested cryptographic protocols for privacy-preserving location proximity testing. Yet general and precise location proximity testing has been out of reach for the current research. A major roadblock has been the requirement by much of the previous work that for proximity testing between Alice and Bob both must be present online. This requirement is not problematic for one-to-one proximity testing but it does not generalize to one-to-many testing. Indeed, in settings like ridesharing, it is desirable to match against ride preferences of all users, not necessarily ones that are currently online.

This paper proposes a novel privacy-preserving proximity testing protocol where, after providing some data about its location, one party can go offline (nap) during the proximity testing execution, without undermining user privacy. We thus break away from the limitation of much of the previous work where the parties must be online and interact directly to each other to retain user privacy. Our basic protocol achieves privacy against semi-honest parties and can be upgraded to full security (against malicious parties) in a straight forward way using advanced cryptographic tools. Finally, we reduce the responding client overhead from quadratic (in the proximity radius parameter) to constant, compared to the previous research. Analysis and performance experiments with an implementation confirm our findings.

1.1 Introduction

We use more and more sophisticated smart phones, wear smart watches, watch programs on smart TVs, equip our homes with smart tools to regulate the temperature, light switches and so on.

Location Based Services. As we digitalize our lives and increasingly rely on smart devices and services, *location based services (LBS)* are among the most widely employed ones. These range from simple automation like "switch my phone to silent mode if my location is office", to more advanced services that involve interaction with other parties, as in "find nearby coffee shops", "find nearby friends", or "find a ride".

Preserving Privacy for Location Based Services. Current practices rely on full trust to the LBS providers: users share their locations with the providers who manipulate location data on behalf of the users. For example, social apps Facebook and Tinder require access to user location in order to check if other users are nearby. Unfortunately, location and user data has been often breached by the LBS providers [10]. The ridesharing app Uber has been reported to violate location privacy of users by stalking journalists, VIPs, and ex-partners [6], as well as ex-filtrating user location information from its competitors [16]. This raises privacy concerns over the current practices.

Privacy-Preserving Proximity Testing. To address these concerns previous research has suggested cryptographic protocols for privacy-preserving location services. The focus of this paper is on the problem of *proximity testing*, the problem of determining if two parties are nearby without revealing any other information about their location. Proximity testing is a useful ingredient for many LBS. For example, ridesharing services are often based on determining the proximity of ride endpoints [17]. There is extensive literature (discussed in Section 1.6) on the problem of proximity testing [13, 19, 20, 24, 28, 29, 35, 36, 37, 38, 39, 41].

Generality and Precision of Proximity Testing. Yet general and precise location proximity testing has been out of reach for the current research. A major roadblock has been the requirement that proximity testing between Alice and Bob is only possible in a pairwise fashion and both must be present online. As a consequence, Alice cannot have a single query answered with respect to multiple Bobs, and nor can she is able to check proximity with respect to Bob's preferences unless Bob is online.

The popular ridesharing service BlaBlaCar [3] (currently implemented as a full-trust service) is an excellent fit to illustrate our goals. This service targets intercity rides which users plan in advance. It is an important requirement that users might go off-line after submitting their preferences. The goal is to find rides that start and end at about the same location. Bob (there can be many Bobs) submits the endpoints of a desired ride to the service and goes offline (napping). At a later point Alice queries the service for proximity testing of her endpoints with Bob's. A key requirement is that Alice should be able to perform a one-to-many proximity query, against all Bobs, and compute answer even if Bob is offline. Unfortunately, the vast majority of the previous work [13, 19, 20, 24, 29, 35, 36, 37, 38, 39, 41] fall short of addressing this requirement.

Another key requirement for our work is precision. A large body of prior approaches [13, 26, 28, 29, 30, 38, 39, 41] resort to grid-based approximations where the proximity problem is reduced to the problem of checking whether the parties are located in the same cell on the grid. Unfortunately, grid-based proximity suffers from both false positives and negatives and can be exploited when crossing cell boundaries [9]. In contrast, our work targets precise proximity testing.

This paper addresses privacy-preserving proximity testing with respect to napping parties. Beyond the described offline functionality and precision we summarize the requirements for our solution as follows: (1) security, in the sense that Alice may not learn anything else about Bob's location other than the proximity; Bob should not learn anything about Alice's location; and the service provider should not learn anything about Alice's or Bob's locations; (2) generality, in the sense that the protocol should allow for one-to-many matching without demanding all users to be online; (3) precision, in the sense of a reliable matching method, not an approximate one; (2) lightweight client computation, in the sense of offloading the bulk of work to intermediate servers. We further articulate on these goals in Section 1.2.

Contributions. This paper proposes OLIC (OffLine Inner-Circle), a novel protocol for proximity testing (Section 1.4). We break away from the limita-

tion of much of the previous work where the parties must be online. Drawing on Hallgren et al.'s two-party protocol InnerCircle [20] we propose a novel protocol for proximity testing that utilizes two non-colluding servers. One server is used to blind Bob's location in such a way that the other server can unblind it for any Alice. Once they have uploaded their locations users in our protocol can go offline and retrieve the match outcome the next time they are online.

In line with our goals, we guarantee security with respect to semi-honest parties, proving that the only location information leaked by the protocol is the result of the proximity test revealed to Alice (Section 1.4.2). We then show how to generically mitigate malicious (yet non-colluding) servers by means of zero knowledge proofs and multi-key homomorphic signatures (Section 1.4.3). Generality in the number of users follows from the fact that users do not need to be online in a pairwise fashion, as a single user can query proximity against the encrypted preferences of the other users. We leverage InnerCircle to preserve the precision, avoiding to approximate proximity information by grids or introducing noise. Finally, OLIC offloads the bulk of work from Bob to the servers, thus reducing Bob's computation and communication costs from quadratic (in the proximity radius parameter) to constant. We note, that while InnerCircle can also be trivially augmented with an extra server to offload Bob's computations to, this will add extra security assumptions and make InnerCircle less applicable in practice. OLIC, on the other hand, already requires the servers for Bob to submit his data to, and we get offloading for free. On Alice's side, the computation and communication costs stay unchanged. We develop a proof of concept implementation of OLIC and compare it with InnerCircle. Our performance experiments confirm the aforementioned gains (Section 1.5).

On the 2 Non-Colluding Servers Assumption. We consider this assumption to be realistic, in the sense that it significantly improves on the current practices of a single full-trust server as in BlaBlaCar, while at the same time being compatible with common assumptions in practical cryptographic privacy-preserving systems. For example, Sharemind [37] requires three non-colluding servers for its multi-party computation system based on 3-party additive secret sharing. To the best of our knowledge, OLIC represents the first 2-server solution to perform proximity testing against napping users in ridesharing scenarios, where privacy, precision and efficiency are all mandatory goals. Notably, achieving privacy using a single server is known to be impossible [16]. Indeed, if Bob was to submit his data to only one server (instead of sharing it between two, as done in OLIC), then the server could

query itself on this data and learn Bob's location via trilateration attack.

1.2 Modeling Private Proximity Testing Using Two Servers

The goal of private proximity testing is to enable one entity, that we will call Alice, to find out whether another entity, Bob, is within a certain distance of her. Note that the functionality is asymmetric: only Alice learns whether Bob lies in her proximity. The proximity test should be performed without revealing any additional information regarding the precise locations to one another, or to any third party.

We are interested in designing a protocol to perform privacy preserving proximity testing exploiting the existence of two servers. For convenience, we name the four parties involved in the protocol: Alice, Bob, Server_1, and Server_2.

Our Setting. We consider a multi party computation protocol for four parties (Alice, Bob, Server_1, and Server_2) that computes the proximity of Alice's and Bob's inputs in a private way and satisfies the following three constraints:

(C-1) Alice does not need to know Bob's identity before starting the test, nor the parties need to share any common secret;

(C-2) Bob needs to be online only to update his location data. In particular, Bob can 'nap' during the actual protocol execution.

(C-3) The protocol is executed with the help of two servers.

In detail, constraint **(C-1)** ensures that Alice can look for a match in the database without necessarily targeting a specific user. This may be relevant in situations where one wants to check the availability of a ride 'near by', instead of searching if a specific cab is at reach and aligns with our generality goal towards one-to-many matching. Constraint **(C-2)** is rarely considered in the literature. The two most common settings in the literature are either to have Alice and Bob communicate directly to one another (which implies that either the two parties need to be online at the same time) [20], or to rely on a single server, which may lead either to 'hiccup' executions (lagging until the other party rejoins online) [29] or to the need for a trusted server. In order to ensure a smooth executions even with a napping Bob and to reduce the trust in one single server, we make use of two servers to store Bob's contribution

to the proximity test, that is constraint **(C-3)**. This aligns with our goal of lightweight client computation. We remark that, for a napping Bob, privacy is lost if we use a single server [16].

Finally, unlike [29] we do not require the existence of a social network among system users, to determine who trusts whom, nor do we rely on shared secret keys among users.

Formalizing 'Napping'. We formalize the requirement that 'Bob may be napping' during the protocol execution in the following way. It is possible to split the protocol in two sequential phases. In the first phase, Bob is required to be online and to upload data to the two servers. In the second phase Alice comes online and perform her proximity test query to one server, that we call Server_1. The servers communicate with one another to run the protocol execution, and finally Server_2 returns the result to Alice.

Ideal Functionality. We adopt an ideal functionality that is very close to the one in [20]: if Alice and Bob are within a distance r^2 of each other, the protocol outputs 1 (to Alice), otherwise it outputs 0 (to Alice). Figure 1.1 depicts this behavior. Alice and Bob are the only parties giving inputs to the protocol. Server_1 and Server_2 do not give any input nor receive any output. This approach aligns with our goal towards precision and a reliable matching method, and brakes away from approximate approaches. For simplicity, we assume the threshold value r^2 to be set a priori, but our protocol accommodates for Alice (or Bob) to choose this value.

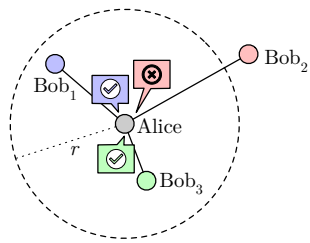


Figure 1.1: Figurative representation of the functionality implemented by privacy-preserving location proximity.

Practical efficiency. Finally, we are interested in solutions that are efficient and run in reasonable time on commodity devices (e.g., laptop computers, smartphones). Concretely, we aim at reducing the computational burden of the clients—Alice and Bob—so that their algorithms can run in just a few seconds, and can in the worst case match the performance of InnerCircle.

Attacker Model. We make the following assumptions on the four parties involved in the protocol:

(A-1) Server_1, Server_2 are not colluding;

(A-2) All parties (Alice, Bob, Server_1, and Server_2) are honest but curious, i.e., they meticulously follow the protocol but may log all messages and attempt to infer some further knowledge on the data they see.

In our model any party could be an attacker that tries to extract un-authorized information from the protocol execution. However, we mainly consider attackers that do so without deviating from their expected execution. In Section 1.4.3, we show that it is possible to relax assumption **(A-2)** for the servers and deal with malicious Server_1, Server_2 (still not colluding). In this case, the servers may misbehave and output a different result than what expected from the protocol execution. However, we do not let the two server collude and intentionally share information. While we can handle malicious servers, we cannot tolerate a malicious Alice or Bob. Detecting attacks such as Alice or Bob providing fake coordinates, or Alice trilaterating Bob to learn his location,is outside our scope and should be addressed by different means. We regard such attacks as orthogonal to our contribution and suggest to mitigate them by employing tamper-resistant location devices or location tags techniques [29].

Privacy. Our definition of privacy goes along the lines of [29]. Briefly, a protocol is private if it reveals no information other than the output of the computation, i.e., it has the same behavior as the ideal functionality. To show the privacy of a protocol we adopt the standard definition of simulation based security for deterministic functionalities [12]. Concretely, we will argue the indistinguishably between a real world and an ideal world execution of our protocol, assuming that the building blocks are secure, there is no collusion among any set of parties and all parties are honest but curious. This means that none of the parties involved in the OLIC protocol can deduce anything they are not supposed to know by observing the messages they receive during protocol execution.

General limitations. Proximity testing can be done in a private way only for a limited amount of requests [20, 29] in order to avoid trilateration attacks [43]. Practical techniques to limit the number of requests to proximity services are available [33]. Asymmetric proximity testing is suitable for checking for nearby people, devices and services. It is known [41] that the asymmetric setting not directly generalize to mutual location proximity testing unless an honest party runs the protocol twice, with swapped roles but using the same location as input).

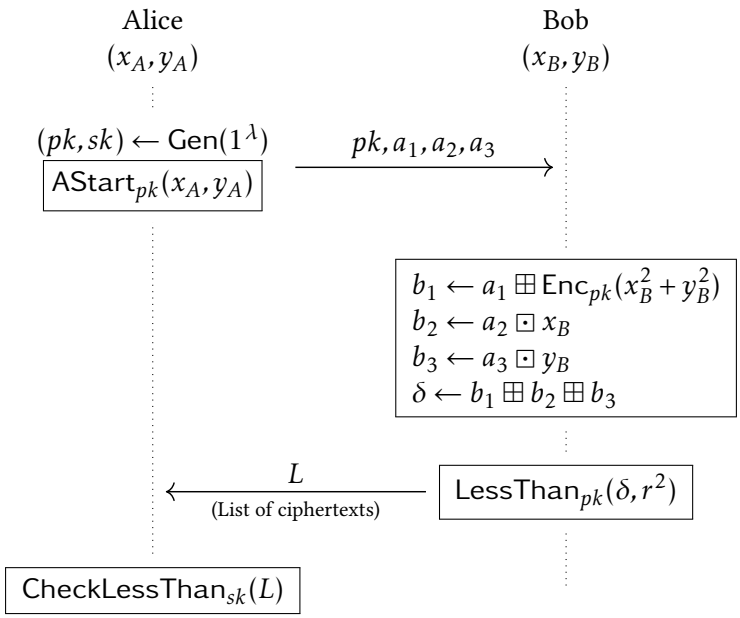


Figure 1.2: Diagram of the InnerCircle protocol.

1.3 Recap of the InnerCircle Protocol

We begin by giving a high-level overview of the InnerCircle protocol by Hallgren et al. [20] and then describe its relation to our work.

The InnerCircle protocol in [20] allows two parties, Alice and Bob, to check whether the Euclidean distance between them is no greater than a chosen value r. The protocol is privacy preserving, i.e., Alice only learns a bit stating whether the Bob lies closer that r or not, while it does not reveal anything to Bob or to any external party. More specifically, Alice and Bob execute an interactive protocol, exchange messages and at the end only Alice receives the answer bit. To let Bob learn the answer as well, one can simply rerun the protocol swapping the roles of Alice and Bob.

More formally, let (x_A, y_A) and (x_B, y_B) respectively denote Alice's and Bob's locations. Throughout the paper, we will use the shorthand $D = (x_A - x_B)^2 + (y_A - y_B)^2$ to denote the squared Euclidean distance between Alice and Bob and δ is an encryption of D. For for any fixed non-negative integer r,

the functionality implemented by the InnerCircle protocol is defined by:

$$F_{\mathsf{InnerCircle}}((x_A, y_A), (x_B, y_B)) = (z, \varepsilon), \text{ where}$$
$$z = \begin{cases} 1, & \text{if } D \leq r^2 \ \ (D = (x_A - x_B)^2 + (y_A - y_B)^2) \\ 0, & \text{otherwise.} \end{cases}$$

InnerCircle is a single round protocol where the first message is from Alice to Bob and the second one is from Bob to Alice (see Figure 1.2).

InnerCircle has three major steps (detailed in Figure 1.3):

Step-1: Alice runs the key generation algorithm to obtain her public key pk and the corresponding sk. She encodes her location (x_A, y_A) into three values and encrypts each of the latter using her pk. The encoding is necessary to enable meaningful multiplication between Alice's coordinates and Bob's as we will see in a moment. Finally, Alice sends her pk and the three ciphertexts to Bob. This step is depicted in Figure 1.3a (the AStart algorithm).

Step-2: Bob uses his location and Alice's pk to homomorphically compute a ciphertext δ that encrypts the value $D = (x_A - x_B)^2 + (y_A - y_B)^2 = (x_A^2 + y_A^2) + (x_B^2 + y_B^2) - 2x_A x_B - 2y_A y_B$. Note that at this point Bob can not learn the value of D, since he never obtains the sk corresponding to Alice's pk. Similarly, if Alice would receive the ciphertext δ, she could decrypt is with sk and learn the exact value of her distance to Bob, which clearly is more information than just a single bit stating whether Bob is *within distance r* from her. To reduce the disclosed information to what is specified in the functionality Bob runs the hiding procedure depicted in Figure 1.3b (the LessThan algorithm). Using this procedure, Bob produces a list L of ciphertexts to return to Alice.

The list L depends on δ in such a way that it contains an encryption of 0 if and only if $D \leq r^2$.

Step-3: After receiving L, Alice decrypts each ciphertext in L using her sk. If among the obtained plaintexts there is a 0 she deduces that $D \leq r^2$. This step is formalized in Figure 1.3c (the CheckLessThan algorithm).

Figure 1.3 contains the definitions of the three procedures used in InnerCircle that will carry on to our OLIC protocol. Below we describe the major ones (LessThan and CheckLessThan) in more detail.

The procedure LessThan (Figure 1.3b) produces a set of ciphertexts from which Alice can deduce whether $D \leq r^2$ without disclosing the actual value D. The main idea behind this algorithm is that given $\delta = \mathsf{Enc}_{pk}(D)$ Bob can "encode" whether $D = i$ in the value $x \leftarrow \delta \oplus \mathsf{Enc}_{pk}(-i)$, and then "mask" it by multiplying it by a random element $l \leftarrow x \boxdot r$. Observe that if $D - i = 0$ then $\mathsf{Dec}_{sk}(l) = 0$; otherwise if $D-i$ is some invertible element of $\mathcal{M}$ then $\mathsf{Dec}_{sk}(l)$ is uniformly distributed on $\mathcal{M} \setminus \{0\}$. (When we instantiate our OLIC protocol for specific cryptosystem, we will ensure that $D-i$ is either 0 or an invertible element, hence we do not consider here the third case when neither of these holds.) The LessThan procedure terminates with a shuffling of the list, i.e., the elements in L are reorganized in a random order so that the entry index no longer correspond to the value i.

$\mathsf{AStart}_{pk}(x_A, y_A)$

1: $a_1 \leftarrow \mathsf{Enc}_{pk}(x_A^2 + y_A^2)$
2: $a_2 \leftarrow \mathsf{Enc}_{pk}(2x_A)$
3: $a_3 \leftarrow \mathsf{Enc}_{pk}(2y_A)$
4: **return** (a_1, a_2, a_3)

(a) The AStart algorithm.

$\mathsf{LessThan}_{pk}(\delta, r^2)$

1: **for** $i \in \{0 \dots r^2 - 1\}$
2: $\quad x_i \leftarrow \delta \boxplus \mathsf{Enc}_{pk}(-i)$
3: $\quad t_i \leftarrow \mathcal{M} \setminus \{0\}$
4: $\quad l_i \leftarrow x_i \boxdot t_i$
5: $L \leftarrow [l_0, \dots l_{r^2-1}]$
6: **return** $\mathsf{Shuffle}(L)$

(b) The LessThan algorithm.

$\mathsf{CheckLessThan}_{sk}(L)$

1: $[l_0, l_2 \dots l_{r^2-1}] \leftarrow L$
2: **for** $i \in \{0 \dots r^2 - 1\}$
3: $\quad v \leftarrow \mathsf{Dec}_{sk}(l_i)$
4: $\quad$ **if** $v = 0$
5: $\quad\quad$ **return** 1
6: **return** 0

(c) The CheckLessThan algorithm.

Figure 1.3: The core algorithms in the InnerCircle protocol.

The procedure CheckLessThan (called inProx in [20] depicted in Figure 1.3c. This procedure takes as input the list L output by LessThan and decrypts one by one all of its components. The algorithm returns 1 if and only if there is a list element that decrypts to 0. This tells Alice whether $D = i$ for some $i < r^2$. We remark that Alice cannot infer any additional information, in particular she cannot extract the exact value i for which the equality holds. In the LessThan procedure Bob computes such "encodings" l for all $i \in \{0 \dots r^2\}$ and accumulates them in a list. So if $D \in \{0 \dots r^2 - 1\}$ is the case, then one of the list elements will decrypt to zero and others will decrypt to uniformly random $\mathcal{M} \setminus \{0\}$ elements, otherwise all of the list elements will decrypt to random nonzero elements. If the cryptosystem allowed multiplying encrypted values, Bob could compute the product of the "encodings" instead of collecting them into a list and send

only a single plaintext, but the cryptosystem used here is only additively homomorphic and does not allow multiplications.

1.4 Private Location Proximity Testing with Napping Bob

Notation. We denote an empty string by ε, and a list of values by $[\cdot]$ and the set of plaintexts (of an encryption scheme) by $\mathcal{M}$. We order parties alphabetically, protocols' inputs and outputs follow the order of the parties. Finally, we denote the computationally indistinguishablity of two distributions D_1, D_2 by $D_1 \stackrel{c}{\equiv} D_2$.

1.4.1 OLIC: Description of the Protocol

In what follows, we describe our protocol for privacy-preserving location proximity testing with a napping Bob. We name this protocol OLIC (for OffLine InnerCircle) to stress its connection with the InnerCircle protocol and the fact that it can run while Bob is offline. More specifically, instead of exchanging messages directly with one another (as in InnerCircle), in OLIC Alice and Bob communicate with (and through) two servers.

The Ideal Functionality of OLIC. At a high level, OLIC takes as input two locations (x_A, y_A) and (x_B, y_B), and a radius value r; and returns 1 if the Euclidean distance between the locations is less than or equal to r, and 0 otherwise. We perform this test with inputs provided by two parties, Alice and Bob, and the help of two servers, Server_1 and Server_2. Formally, our ideal functionality has the same input and output behavior as InnerCircle for Alice and Bob, but it additionally has two servers whose inputs and outputs are empty strings ε.

$$F_{\text{OLIC}}((x_A, y_A), (x_B, y_B), \varepsilon, \varepsilon) = (\texttt{res}, \varepsilon, \varepsilon, \varepsilon),$$
$$\text{where } \texttt{res} = \begin{cases} 1, & \text{if } (x_A - x_B)^2 + (y_A - y_B)^2 \leq r^2 \\ 0, & \text{otherwise.} \end{cases} \tag{1.1}$$

In our protocol we restrict the input/output spaces of the ideal functionality of Equation 1.4.1 to values that are meaningful to our primitives. In other words, we require that the values x_A, y_A, x_B, y_B are admissible plaintext (for the encryption scheme employed in OLIC), and that the following values are invertible (in the ring of plaintext) $x_B, y_B, D - i$ for $i \in \{0, \ldots, r^2 - 1\}$ and

$D = (x_A - x_B)^2 + (y_A - y_B)^2$. We will denote the set of all suitable inputs as $\mathcal{S}_\lambda \subseteq \mathcal{M}$.

Figure 1.4 depicts the flow of the OLIC protocol. Figure 1.5 contains the detailed description of the procedures called in Figure 1.4.

Step-0: Alice, Server_1, and Server_2 independently generate their keys. Alice sends her pk to Server_1 and Server_2; Server_1 and Server_2 send their respective public keys pk_1, pk_2 to Bob.

Step-1: At any point in time, Bob encodes his location using the BStart algorithm (Figure 1.5a), and sends its blinded coordinates to Server_1, and the corresponding blinding factors to Server_2.

Step-2: At any point in time Alice runs AStart (Figure 1.3a) and sends her ciphertexts to Server_1.

Step-3: once Server_1 collects Alice's and Bob's data it can proceed with computing 3 addends useful to obtain the (squared) Euclidean distance between their locations. To do so, Server_1 runs CompTerms (Figure 1.5b), and obtains:

$$\begin{aligned} c_1 &= \mathsf{Enc}_{pk}(x_A^2 + y_A^2 + (x_B^2 + y_B^2 + r_1)) \\ c_2 &= \mathsf{Enc}_{pk}(2x_A + (x_B + r_2)) \\ c_3 &= \mathsf{Enc}_{pk}(2y_A + (y_B + r_3)) \end{aligned}$$

Server_1 sends the above three ciphertexts to Server_2.

Step-4: Server_2 runs UnBlind (Figure 1.5c) to remove Bob's blinding factors from c_1, c_2, c_3 and obtains the encrypted (squared) Euclidean distance between Alice and Bob:

$$\begin{aligned} \delta &= c_1 \boxplus \mathsf{Enc}_{pk}(-r_1) \boxplus c_2 \boxdot \mathsf{Enc}_{pk}(r_2^{-1}) \boxplus c_3 \boxdot \mathsf{Enc}_{pk}(r_3^{-1}) \\ &= \mathsf{Enc}_{pk}((x_A - x_B)^2 + (y_A - y_B)^2) \end{aligned}$$

Then Server_2 uses δ and the radius value r to run LessThan (as in InnerCircle, Figure 1.3b), obtains the list of ciphertexts L and returns L to Alice.

Step-5: Alice runs CheckLessThan (Figure 1.3c) to learn whether $D \leq r^2$ in the same way as done in InnerCircle.

The correctness of OLIC follows from the correctness of InnerCircle and a straightforward computation of the input-output of BStart, CompTerms, and UnBlind.

Figure 1.4: Overview of the message flow of our OLIC protocol. The message exchanges are grouped to reduce vertical space; in a real execution of the protocol Bob may submit at any time before CompTerms is run.

$\mathsf{BStart}(pk_1, pk_2, x_B, y_B)$

1: $r_1, \xleftarrow{\$} M$
2: $r_2, r_3 \xleftarrow{\$} M^*$
3: $b_1 \leftarrow \mathsf{Enc}_{pk_1}(x_B^2 + y_B^2 + r_1)$
4: $b_2 \leftarrow \mathsf{Enc}_{pk_1}(x_B r_2)$
5: $b_3 \leftarrow \mathsf{Enc}_{pk_1}(y_B r_3)$
6: $b_1' \leftarrow \mathsf{Enc}_{pk_2}(-r_1)$
7: $b_2' \leftarrow \mathsf{Enc}_{pk_2}(r_2^{-1})$
8: $b_3' \leftarrow \mathsf{Enc}_{pk_2}(r_3^{-1})$
9: **return** $(b_1, b_2, b_3, b_1', b_2', b_3')$

(a) The BStart algorithm.

$\mathsf{CompTerms}_{sk_1}(pk, a_1, a_2, a_3, b_1, b_2, b_3)$

1: $tmp_1 \leftarrow \mathsf{Dec}_{sk_1}(b_1)$
2: $tmp_2 \leftarrow \mathsf{Dec}_{sk_1}(b_2)$
3: $tmp_3 \leftarrow \mathsf{Dec}_{sk_1}(b_3)$
4: $c_1 \leftarrow a_1 \boxplus \mathsf{Enc}_{pk}(tmp_1)$
5: $c_2 \leftarrow a_2 \boxdot tmp_2$
6: $c_3 \leftarrow a_3 \boxdot tmp_3$
7: **return** (c_1, c_2, c_3)

(b) The CompTerms algorithm.

$\mathsf{UnBlind}_{sk_2}(pk, c_1, c_2, c_3, b_1', b_2', b_3')$

1: $tmp_1 \leftarrow \mathsf{Dec}_{sk_2}(b_1')$
2: $tmp_2 \leftarrow \mathsf{Dec}_{sk_2}(b_2')$
3: $tmp_3 \leftarrow \mathsf{Dec}_{sk_2}(b_3')$
4: $d_1 \leftarrow c_1 \boxplus \mathsf{Enc}_{pk}(tmp)_1$
5: $d_2 \leftarrow c_2 \boxdot tmp_2$
6: $d_3 \leftarrow c_3 \boxdot tmp_3$
7: **return** $\delta \leftarrow d_1 \boxplus d_2 \boxplus d_3$

(c) The UnBlind algorithm.

Figure 1.5: The new subroutines in OLIC.

1.4.2 OLIC: Privacy of the Protocol

We prove that our OLIC provides privacy against semi-honest adversaries. We do so by showing an efficient simulator for each party involved in the protocol and then arguing that each simulator's output is indistinguishable from the view of respective party. More details on the cryptographic primitives and their security model can be found in Appendix 1.A, or in [4, 12].

Theorem 1. *If the homomorphic encryption scheme*

$$\mathsf{HE} = (\mathsf{Gen}, \mathsf{Enc}, \mathsf{Dec}, \mathsf{Eval})$$

used in OLIC *is IND-CPA secure and function private then* OLIC *securely realizes the privacy-preserving location proximity testing functionality (in Equation*

1.4.1) against semi-honest adversaries.

We prove Theorem 1 by showing four simulators whose outputs are computationally indistinguishable by nonuniform algorithms from the views of the parties. Concretely, we will prove four lemmas, each dealing with a different party. Our simulators have the following structure:

$$\mathsf{simParty}(\text{Party's input}, \text{Party's output}) = (\text{Party's view}).$$

Moreover, will append a $'$ symbol to denote the algorithms which return their random bits in addition to their usual result, e.g., if the key-generation algorithm $(pk, sk) \leftarrow \mathsf{Gen}(1^\lambda)$ returns a pair of keys, then its counterpart $(pk, sk, r) \leftarrow \mathsf{Gen}'(1^\lambda)$ returns the same output together with the string of random bits r used to generate keys. Without loss of generality we will assume that all four parties use exactly $p(\lambda)$ random bits in one run of the OLIC protocol, for some polynomial $p(\cdot)$.

Lemma 1. *For all* suitable *inputs* $(x_A, y_A, x_B, y_B) \in \mathcal{S}_\lambda$ *for the* OLIC *protocol, there exists a simulator* simAlice *such that:*

$$\mathsf{simAlice}((x_A, y_A), \mathtt{res}) \stackrel{c}{\equiv} view^{\mathsf{OLIC}}_{Alice}((x_A, y_A), (x_B, y_B), \varepsilon, \varepsilon),$$
$$\text{where } \mathtt{res} = \begin{cases} 1, & \text{if } (x_A - x_B)^2 + (y_A - y_B)^2 \leq r^2 \\ 0, & \text{otherwise.} \end{cases}$$

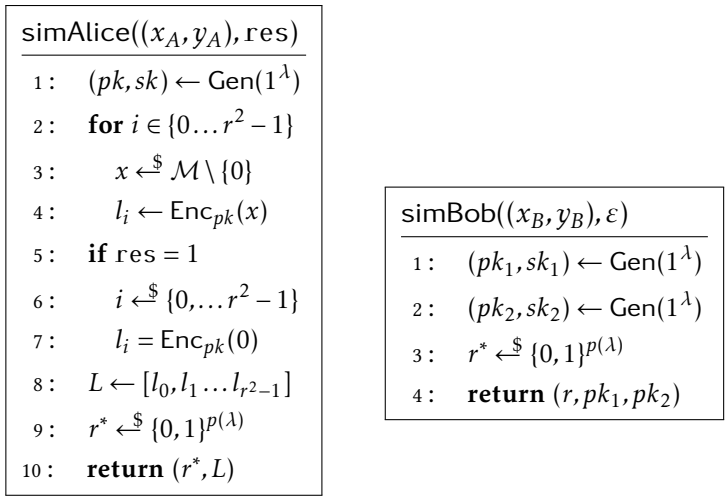
Proof. Alice receives only one message: the list L. The distribution of L is defined by the bit res: when $\mathtt{res} = 0$ the L is a list of encryptions of random nonzero elements of $\mathcal{M}$, otherwise one of its elements (the position of which is chosen uniformly at random) contains an encryption of zero.

This is exactly how simAlice (defined in Figure 1.6a) creates the list L according to res. It is immediate to see that the distributions in Lemma 1 are not only computationally indistinguishable, but actually equal (perfectly indistinguishable). □

Lemma 2. *For all* suitable *inputs* $(x_A, y_A, x_B, y_B) \in \mathcal{S}_\lambda$ *there exists a simulator* simBob *such that:*

$$\mathsf{simBob}((x_A, y_A), \varepsilon) \stackrel{c}{\equiv} view^{\mathsf{OLIC}}_{Bob}((x_A, y_A), (x_B, y_B), \varepsilon, \varepsilon).$$

Proof. Apart from a sequence of random bits, Bob's view consists of two public keys which the servers freshly generate before sending them to him. This is exactly how simBob (in Figure 1.6b) obtains the keys it returns, so the two distributions are not only computationally indistinguishable, but also are equal. □



(a) The simulator for Alice. **(b)** The simulator for Bob.

Figure 1.6: Simulators for Alice and Bob.

Lemma 3. *For all* suitable *inputs* $(x_A, y_A, x_B, y_B) \in \mathcal{S}_\lambda$ *here exists a simulator* $\mathsf{simS1}$ *is such that:* $\mathsf{simS1}(\varepsilon, \varepsilon) \stackrel{c}{\equiv} view^{\mathsf{OLIC}}_{\mathrm{Server}_1}((x_A, y_A), (x_B, y_B), \varepsilon, \varepsilon)$.

Proof. We show that the outputs produced by the two algorithms from Figure 1.7a and 1.7b are computationally indistinguishable by non uniform algorithms (assuming HE is IND-CPA and circuit private). First, we observe that the outputs of both algorithms consist of three parts which are generated independently: (pk, a_1, a_2, a_3), $(r, b_1, b_2, b3)$, r^*. It is easy to see that the last two parts are generated in the same way by both algorithms, so the distributions are actually equal on these parts. The first part (pk, a_1, a_2, a_3) of both distributions are indistinguishable because of the IND-CPA property of used encryption scheme. □

Lemma 4. *For all* suitable *inputs* $(x_A, y_A, x_B, y_B) \in \mathcal{S}_\lambda$ *there exists a simulator* $\mathsf{simS2}$ *such that:* $\mathsf{simS2}(\varepsilon, \varepsilon) \stackrel{c}{\equiv} view^{\mathsf{OLIC}}_{\mathrm{Server}_2}((x_A, y_A), (x_B, y_B), \varepsilon, \varepsilon)$.

Proof. Like in the previous proof, we need to prove here that the outputs of the functions from Figures 1.8a and 1.8b are indistinguishable. Observe that the (rr^*, b'_1, b'_2, b'_3) is generated independently from the rest of the output in both algorithms, and this part is generated in the same way in both places, so the both distributions are equal on this part. The rest of the outputs of the two algorithms, namely (pk, c_1, c_2, c_3), are indistinguishable from one another because of the IND-CPA property of the used cryptosystem. □

$\mathsf{simS1}(\varepsilon, \varepsilon)$

1 : $(pk, sk) \leftarrow \mathsf{Gen}(1^\lambda)$
2 : $a_1 \leftarrow \mathsf{Enc}_{pk}(0)$
3 : $a_2 \leftarrow \mathsf{Enc}_{pk}(0)$
4 : $a_3 \leftarrow \mathsf{Enc}_{pk}(0)$
5 : $(pk_1, sk_1, r) \leftarrow \mathsf{Gen}'(1^\lambda)$
6 : $\beta_1 \xleftarrow{\$} \mathcal{M}$
7 : $\beta_2 \xleftarrow{\$} \mathcal{M}^*$
8 : $\beta_3 \xleftarrow{\$} \mathcal{M}^*$
9 : $b_1 \leftarrow \mathsf{Enc}_{pk_1}(\beta_1)$
10 : $b_2 \leftarrow \mathsf{Enc}_{pk_1}(\beta_2)$
11 : $b_3 \leftarrow \mathsf{Enc}_{pk_1}(\beta_3)$
12 : $r^* \xleftarrow{\$} \{0,1\}^{p(\lambda)-|r|}$
13 : **return** $(rr^*, pk, a_1, a_2, a_3, b_1, b_2, b_3)$

(a) The simulator for Server_1.

$view^{\mathsf{OLIC}}_{\mathsf{Server}_1}((x_A, y_A), (x_B, y_B), \varepsilon, \varepsilon)$

1 : $(pk, sk) \leftarrow \mathsf{Gen}(1^\lambda)$
2 : $(pk_1, sk_1, r) \leftarrow \mathsf{Gen}'(1^\lambda)$
3 : $(a_1, a_2, a_3) \leftarrow \mathsf{AStart}_{pk}(x_A, y_A)$
4 : $r_1, \xleftarrow{\$} M$
5 : $r_2, r_3 \xleftarrow{\$} M^*$
6 : $b_1 \leftarrow \mathsf{Enc}_{pk_1}(x_B^2 + y_B^2 + r_1)$
7 : $b_2 \leftarrow \mathsf{Enc}_{pk_1}(x_B r_2)$
8 : $b_3 \leftarrow \mathsf{Enc}_{pk_1}(y_B r_3)$
9 : $r^* \xleftarrow{\$} \{0,1\}^{p(\lambda)-|r|}$
10 : **return** $(rr^*, pk, a_1, a_2, a_3, b_1, b_2, b_3)$

(b) The view of Server_1.

Figure 1.7: The simulator and view of Server_1 in OLIC.

Privacy Remark on Bob. In case the two servers collude, Bob looses privacy, in the sense that Server_1 and Server_2 together can recover Bob's location (by unblinding the tmp_i in CompTerms, Figure 1.5b). However Alice retains her privacy even in case of colluding servers (thanks to the security of the encryption scheme).

1.4.3 Security Against Malicious Servers

In Section 1.4.2 we proved that OLIC is secure against semi-honest adversaries. In what follows, we show how to achieve security against malicious servers as well. We do it in a generic way, assuming two *not*-colluding servers, a suitable non-interactive zero knowledge proof system (NIZK) and employing a fairly novel cryptographic primitive called multi-key homomorphic signatures (MKHS) [1, 10]. The proposed maliciously secure version of OLIC is currently more of a feasibility result rather than a concrete instantiation: to the best of our knowledge there is no combination of MKHS and NIZK that would fit our needs.

Homomorphic signatures [5, 14] enable a signer to authenticate their data

$\mathsf{simS2}(\varepsilon, \varepsilon)$

$(pk_1, sk_1, r) \leftarrow \mathsf{Gen}'(1^\lambda)$
$\beta_1 \xleftarrow{\$} \mathcal{M}$
$\beta_2 \xleftarrow{\$} \mathcal{M}^*$
$\beta_3 \xleftarrow{\$} \mathcal{M}^*$
$b'_1 \leftarrow \mathsf{Enc}_{pk_s}(\beta_1)$
$b'_2 \leftarrow \mathsf{Enc}_{pk_s}(\beta_2)$
$b'_3 \leftarrow \mathsf{Enc}_{pk_s}(\beta_3)$
$(pk, sk) \leftarrow \mathsf{Gen}(1^\lambda)$
$c_1 \leftarrow \mathsf{Enc}_{pk}(0)$
$c_2 \leftarrow \mathsf{Enc}_{pk}(0)$
$c_3 \leftarrow \mathsf{Enc}_{pk}(0)$
$r^* \xleftarrow{\$} \{0,1\}^{p(\lambda)-|r|}$
return $(rr^*, pk, b'_1, b'_2, b'_3, c_1, c_2, c_3)$

(a) The simulator for Server_2.

$view^{\mathsf{OLIC}}_{\mathsf{Server}_2}((x_A, y_A), (x_B, y_B), \varepsilon, \varepsilon)$

$(pk, sk) \leftarrow \mathsf{Gen}(1^\lambda)$
$(pk_2, sk_2, r) \leftarrow \mathsf{Gen}'(1^\lambda)$
$r_1, \xleftarrow{\$} M$
$r_2, r_3 \xleftarrow{\$} M^*$
$b'_1 \leftarrow \mathsf{Enc}_{pk_2}(-r_1)$
$b'_2 \leftarrow \mathsf{Enc}_{pk_2}(r_2^{-1})$
$b'_3 \leftarrow \mathsf{Enc}_{pk_2}(r_3^{-1})$
$c_1 \leftarrow \mathsf{Enc}_{pk}(x_A^2 + y_A^2 + x_B^2 + y_B^2 + r_1)$
$c_2 \leftarrow \mathsf{Enc}_{pk}(2x_A x_B r_2)$
$c_3 \leftarrow \mathsf{Enc}_{pk}(2y_A y_B r_3)$
$r^* \xleftarrow{\$} \{0,1\}^{p(\lambda)-|r|}$
return $(rr^*, pk, b'_1, b'_2, b'_3, c_1, c_2, c_3)$

(b) The view of Server_2.

Figure 1.8: The simulator and view of Server_2 in OLIC.

in such a way that any third party can homomorphically compute on it and obtain (1) the result of the computation, and (2) a signature vouching for the correctness of the latter result. In addition to what we just described, MKHS make it possible to authenticate computation on data signed by multiple sources. Notably, homomorphic signatures and MKHS can be used to efficiently verify that a computation has been carried out in the correct way on the desired data without need to access the original data [10]. This property is what we leverage to make OLIC secure against malicious servers.

At a high level, our proposed solution to mitigate malicious servers works as follows. Alice and Bob hold distinct secret signing keys, sk_A and sk_B respectively, and sign their ciphertexts before uploading them to the servers. In detail, using the notation in Figure 1.4, Alice sends to Server_1 the three messages (ciphertexts) a_1, a_2, a_3 along with their respective signatures $\sigma_i^A \leftarrow \mathsf{MKHS.Sign}(\mathsf{sk}_A, a_i, \ell_i)$, where ℓ_i is an opportune label.[1] Bob acts similarly. Server_1 computes f_i (the circuit corresponding to the computation in CompTerms on the i-th input) on each ciphertext and each signature, i.e., $c_i \leftarrow f(a_i, b_i)$

[1] More details on labels at the end of the section.

and $\sigma_i' \leftarrow \mathsf{MKHS.Eval}(f, \{\mathsf{pk}_A, \mathsf{pk}_B\}, \sigma_i^A, \sigma_i^B)$. The unforgeability of MKHS ensures that each multi-key signature σ_i' acts as a proof that the respective ciphertext c_i' has been computed correctly (i.e., using f on the desired inputs[2]). Server_2 can be convinced of this by checking if $\mathsf{MKHS.Verif}(f, \{\ell_j\}, \{\mathsf{pk}_A, \mathsf{pk}_B\}, c_i', \sigma_i')$ returns 1. If so, Server_2 proceeds and computes the value δ (and its multi-key signature σ) by evaluating the circuit g corresponding to the function UnBlind. As remarked in Section 1.4.2, privacy demands that the random coefficients, x_i:s, involved in the LessThan procedure are not leaked to Alice. However, without the x_i:s Alice cannot run the MKHS verification (as this randomness should be hardwired in the circuit h corresponding to the function LessThan). To overcome this obstacle we propose to employ a zero knowledge proof system. In this way Server_2 can state that it knows undisclosed values x_i:s such that the output data (the list $L \leftarrow [l_1, \ldots, l_{r^2-1}]$) passes the MKHS verification on the computations dependent on x_i:s. This can be achieved by interpreting the MKHS verification algorithm as a circuit v with inputs x_i (and l_i).

Security Considerations. To guarantee security we need the MKHS to be context hiding (e.g., [37], to prevent leakage of information between Server_1 and Server_2); unforgeable (in the homomorphic sense [10]); and the final proof to be zero knowledge. Implementing this solution would equip OLIC with a quite advanced and heavy machinery that enables Alice to detect malicious behaviors from the servers.

A Caveat on Labels. There is final caveat on data labels [10] needed for the MKHS schemes. We propose to set labels as a string containing the public information: day, time, identity of the user producing the location data, and data type identifier (e.g., $(1, 3)$ to identify the ciphertext b_3, sent by Bob to Server_1, $(2, 1)$ to identify the ciphertext b_1', sent by Bob to Server_2, and $(0, 2)$ to identify Alice's ciphertext a_2—Alice only sends data to Server_1). We remark that such label information would be retrievable by InnerCircle as well, as in that case Alice knows the identity of her interlocutor (Bob), and the moment (day and time) in which the protocol is run.

1.5 Evaluation

In this section we evaluate the performance of our proposal in three different ways: first we provide asymptotic bounds on time complexity of the

[2]The 'desired' inputs are indicated by the labels, as we discuss momentarily.

algorithms in OLIC; second, we provide bounds on the total communication cost of the protocol; finally we develop a proof-of-concept implementation of OLIC to test its performance (running time and communication cost) and compare it against the InnerCircle protocol. Recall that InnerCircle and OLIC implement the same functionality (privacy-preserving proximity testing), however the former requires Bob to be online during the whole protocol execution while the latter does not.

Parameters. As described in Section 1.4.1, OLIC requires Alice to use an additive homomorphic encryption scheme. However, no special property is needed by the ciphertexts from Bob to the servers and between servers. Our implementation employs the ElGamal cryptosystem over a safe-prime order group for ciphertexts to and from Alice, while for the other messages it uses the Paillier cryptosystem. We refer to this implementation as **(non-EC)**, as it does not rely on any elliptic curive cryptograpy (ECC). In order to provide a fair comparison with its predecessor (InnerCircle), we additionally instantiate OLIC using only ECC cryptosystems **(EC)**, namely Elliptic Curve ElGamal.

We note that additive homomorphic ElGamal relies on the face that a plaintext value m is encoded into a group element as g^m (where g denotes the group generator). In this way, the multiplication of $g^{m_1} \cdot g^{m_2}$ returns an encoding of the addition of the corresponding plaintext values $m_1 + m_2$. In order to 'fully' decrypt a ciphertext, one would have to solve the discrete logarithm problem and recover m from g^m, which should be unfeasible, as this corresponds to the security assumption of the ecryption scheme. Fortunately, this limitation is not crucial for our protocol. In OLIC, indeed, Alice only needs to check whether a ciphertext is an encryption of zero or not (in the CheckLessThan procedure), and this can be done efficiently without fully decrypting the ciphertext. However, we need to keep it into account when implementing OLIC with ECC. Indeed, in OLIC the servers need to actually fully decrypt Bob's ciphertexts in order to work with the corresponding (blinded) plaintexts. We do so by employing a non-homomorphic version of ElGamal based on the curve M383 in **(EC)** and Paillier cryptosystem in **(non-EC)**.

In our evaluation, we ignore the computational cost of initial key-generation as, in real-life scenarios, this process is a one-time set up and is not needed at every run of the protocol.

1.5.1 Asymptotic complexity

Table 1.1 shows both the concrete numbers of cryptographic operations and our asymptotic bounds on the time complexity of the algorithms executed by each party involved in OLIC. We assume that ElGamal and Paillier encryption, decryption and arithmetic operations, are performed in time $\mathcal{O}(\lambda m)$ (assuming binary exponentiation), where λ here is the security parameter and m is the cost of λ-bit interger multiplication — depends on the specific multiplication algorithm used, e.g., $m = O(n \log n \log\log n)$ for Schönhage–Strassen algorithm. This applies to both **(EC)** and **(non-EC)**, although **(EC)** in practice is used with a lot smaller values of λ.

Table 1.1: Concrete number of operations and asymptotic time complexity for each party in OLIC (m is the cost of modular multiplication, λ the security parameter and r the radius). In our implementation Alice decryption is actually an (efficient) zero testing.

Party	Cryptosystem operations	Time bound
Alice	3 encryptions r^2 decryptions	$\mathcal{O}(r^2 \lambda m)$
Bob	6 encryptions	$\mathcal{O}(\lambda m)$
Server_1	3 decryptions, 3 homomorphic operations,	$\mathcal{O}(\lambda m)$
Server_2	3 decryptions, $2r^2 + 5$ arithmetic operations, r^2 encryptions	$\mathcal{O}(r^2 \lambda m)$

Communication cost. The data transmitted among parties during a whole protocol execution amounts to r^2+6 ciphertexts between Alice and the servers and 6 ciphertexts between Bob and servers. Each ciphertext consists of 4λ bits in case of **(EC)**, and 2λ bits in case of **(non-EC)** (for both ElGamal and Paillier). Asymptotically, both cases require $\mathcal{O}(\lambda r^2)$ bit of communication—although **(EC)** is used with a lot smaller λ value in implementation.

1.5.2 Implementation

We developed a prototype implementation of OLIC in Python (available at [32]). The implementation contains all of the procedures shown in pseudocode in Figures 1.3 and 1.5. To ensure a fair compare between OLIC and InnerCircle, we implemented latter in the same environment (following the nomenclature in Figure 1.2).

Our benchmarks for the total communication cost of the OLIC protocol are reported in Figure 1.9. We measured the total execution time of each procedure of both InnerCircle and OLIC. Figure 1.10 shows the outcome of our measurements for values of the proximity radius parameter r ranging from 0 to 100. Detailed values can be found in Appendix 1.B (Table 1.2).

Setup. We used cryptosystems from the cryptographic library bettertimes [18], which was used for benchmarking of the original InnerCircle protocol. The benchmarks were run on Intel Core i7-8700 CPU running at a frequency of 4.4 GHz. For **(non-EC)** we used 2048-bit keys for ElGamal (the same as in InnerCircle [20]), and 2148-bit keys for Paillier to accomodate a modulus larger than the ElGamal modulus (so that any possible message of Bob fits into the Paillier message space). For **(EC)** we used curves Curve25519 for additive homomorphic encryption and M383 for the ciphertexts exchanged among Bob and the servers from the ecc-pycrypto library [7]. We picked these curves because they were available in the library and also because M383 uses a larger modulus and allows us to encrypt the big values from the plaintexts field of Curve25519.

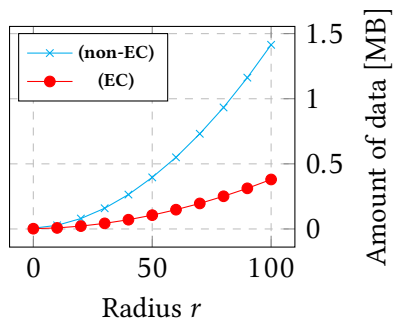


Figure 1.9: Total communication cost of OLIC.

The plaintext ring for **(non-EC)** ElGamal-2048 has at least $|\mathcal{M}| \geq 2^{2047}$ elements, which allows Alice and Bob's points to lie on the grid $\{1, 2 \dots 2^{1023}\}^2$ ensuring that the seqared distance between them never exceeds 2^{2047}. The corresponding plaintext ring size for **(EC)** is $|\mathcal{M}| \geq 2^{251}$ (the group size of Curve25519), and the grid is $\{1, 2 \dots 2^{125}\}^2$. Since Earth equator is $\approx 2^{26}$ meters long, either of the two grids is more than enough to cover any location on Earth with 1 meter precision.

Optimizations. In InnerCircle [20], the authors perform three types of optimizations on the LessThan procedure:

(O-1) Iterating only through those values of $i \in \{0, \dots, r^2 - 1\}$ which can be represented as a sum of two squares, i.e., such i that $\exists\, a, b : i = a^2 + b^2$.

(O-2) Precomputing the ElGamal ciphertexts $\mathsf{Enc}_{pk}(-i)$, and only for those i described in **(O-1)**.

(O-3) Running the procedure in parallel, using 8 threads.

We adopt the optimizations **(O-1)** and **(O-2)** in our implementation as well. Note that **(O-1)** reduces the length of list L (Figure 1.4), as well as the total communication cost. We disregard optimization **(O-3)** since we are interested in the total amount of computations a party needs to do (thus this optimization is not present in our implementations of OLIC and InnerCircle).

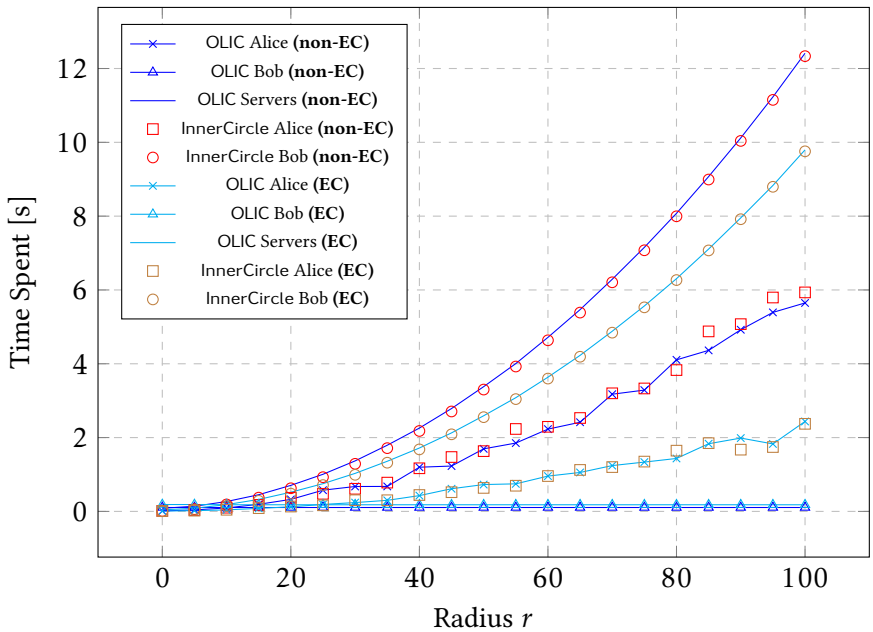


Figure 1.10: Running times of each party in InnerCircle and OLIC for both **(non-EC)** and **(EC)** instantiations. (Reported times are obtained as average of 40 executions.)

1.5.3 Performance Evaluation

Figure 1.10 shows a comparison of the total running time of each party in OLIC versus InnerCircle. One significant advantage of OLIC is that it offloads the execution of the LessThan procedure from Bob to the servers. This is reflected in Figure 1.10, where the running time of Bob is flat in OLIC, in contrast to quadratic as in InnerCircle. Indeed, the combined running time of the two servers in OLIC almost matches the running time of Bob in InnerCircle. Note that the servers have to spend total 10-12 seconds on computations when $r = 100$, which is quite a reasonable amount of time, given

that servers are usually not as resource-constrained as clients, and that the most time-consuming procedure LessThan consists of a single loop which can easily be executed in parallel to achieve even better speed. Finally, we remark that the amount of data being sent (Figure 1.9) between the parties in OLIC is quite moderate.

In case Alice wants to be matched with multiple Bobs, say, with k of them, the amount of computations that she and the servers perform will grow linearly with k. The same applies to the communication cost. Therefore, one can obtain the counterparts of Figures 1.10 and 1.9 for multiple Bobs by simply multiplying all the plots (except Bob's computations) by k.

1.6 Related Work

Zhong et al. [41] present the Louis, Lester and Pierre protocols for location proximity. The Louis protocol uses additively homomorphic encryption to compute the distance between Alice and Bob while it relies on a third party to perform the proximity test. Bob needs to be present online to perform the protocol. 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 resorts to grids and leaks Bob's grid cell distance to Alice.

Hide&Crypt by Freni et al. [13] splits proximity in two steps. Filtering is done between a third party and the initiating principal. The two principals then execute computation to achieve finer granularity. In both steps, the granule 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] presents a centralized protocol where clients map their position 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.

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 protocol utilizes an oblivious server. Parties in this protocol use symmetric encryption, which leads to better performance. However, this requires to have 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 geom-

etry and linear algebra. Nielsen et al. [30] and Kotzanikolaou et al. [26] also propose grid-based solutions.

Šeděnka and Gasti [37] homomorphically compute distances using the UTM projection, ECEF (Earth-Centered Earth-Fixed) coordinates, and the Haversine formula that make it possible to consider the curvature of the Earth. Hallgren et al. [20] introduce InnerCircle for parallizable decentralized proximity testing, using additively homomorphic encryption between two parties that must be online. The MaxPace [19] protocol builds on 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 require Alice and Bob to be online to be able to run the protocol. Järvinen et al. [24] design efficient schemes for Euclidean distance-based privacy-preserving location proximity. They demonstrate performance improvements over InnerCircle. Yet the requirement of the two parties being online applies to their setting as well. Hallgren et al. [17] how to leverage proximity testing for endpoint-based ridesharing, building on the InnerCircle protocol, and compare this method with a method of matching trajectories.

The computational bottle neck of privacy-preserving proximity testing is the input comparison process. Similarly to [20, 33], we rely on homomorphic encryption to compare a private input (the distance between the submitted locations) with a public value (the threshold). Other possible approaches require the use of the arithmetic black box model [6], garbled circuits [25], generic two party computations [12], or oblivious transfer extensions [8].

To summarize, the vast majority [13, 19, 20, 24, 29, 35, 36, 37, 38, 39, 41] of the existing approaches to proximity testings require both parties to be online, thus not being suitable for one-to-many matching. A notable exception to the work above is the C-Hide&Hash protocol by Mascetti et al. [28], which allows one-to-many testing, yet at the price of not computing the precise proximity result but its grid-based approximation. Generally, a large number of approaches [13, 26, 28, 29, 30, 38, 39, 41] resort to grid-based approximations, thus loosing precision of proximity tests.

There is a number of existing works, which consider the problem of computing generic functions in the setting, where clients are not online during the whole execution. Hallevi et al. [16] consider a one-server scenario and show that the notion of security agains semi-honest adversary (which we prove for our protocol) is impossible to achive with one server. Additionally, the model from [16] lets all the parties know each other's public keys, i.e.,

the clients know all the other clients who supply inputs for the protocol—this does not allow one-to-many matching, which we achive in our work. Further works [1, 2, 14, 15, 22] also consider one-server scenarios.

1.7 Conclusions

We have presented OLIC, a protocol for privacy-preserving proximity testing with a napping party. In line with our goals, (1) we achieve privacy with respect to semi-honest parties; (2) we enable matching against offline users which is needed in scenarios like ridesharing; (3) we retain precision, not resorting to grid-based approximations, and (4) we reduce the responding client overhead from quadratic (in the proximity radius parameter) to constant.

Future work avenues include developing a fully-fledged ridesharing system based on our approach, experimenting with scalability, and examining the security and performance in the light of practical security risks for LBS services.

This page intentionally left blank.

Bibliography

[1] D. F. Aranha and E. Pagnin. The simplest multi-key linearly homomorphic signature scheme. In *LATINCRYPT*, pages 280–300. Springer, 2019.

[2] A. Beimel, A. Gabizon, Y. Ishai, E. Kushilevitz, S. Meldgaard, and A. Paskin-Cherniavsky. Non-interactive secure multiparty computation. In J. A. Garay and R. Gennaro, editors, *CRYPTO*, volume 8617 of *LNCS*, pages 387–404. Springer, 2014.

[3] F. Benhamouda, H. Krawczyk, and T. Rabin. Robust non-interactive multiparty computation against constant-size collusion. In J. Katz and H. Shacham, editors, *CRYPTO*, volume 10401 of *LNCS*, pages 391–419. Springer, 2017.

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

[5] D. Boneh and D. M. Freeman. Homomorphic signatures for polynomial functions. In *EUROCRYPT*, pages 149–168. Springer, 2011.

[6] O. Catrina and S. De Hoogh. Improved primitives for secure multiparty integer computation. In *SCN*, pages 182–199. Springer, 2010.

[7] ChangLiu. *Ecc-pycrypto Library*, 2019 (accessed April 14, 2020).

[8] G. Couteau. New protocols for secure equality test and comparison. In *ACNS*, pages 303–320. Springer, 2018.

[9] J. Cuéllar, M. Ochoa, and R. Rios. Indistinguishable regions in geographic privacy. In *SAC*, pages 1463–1469, 2012.

[10] D. Fiore, A. Mitrokotsa, L. Nizzardo, and E. Pagnin. Multi-key homomorphic authenticators. In *ASIACRYPT*, pages 499–530. Springer, 2016.

[11] D. Freni, C. R. Vicente, S. Mascetti, C. Bettini, and C. S. Jensen. Preserving location and absence privacy in geo-social networks. In *CIKM*, pages 309–318, 2010.

[12] J. Garay, B. Schoenmakers, and J. Villegas. Practical and secure solutions for integer comparison. In *PKC*, pages 330–342. Springer, 2007.

[13] C. Gentry, S. Halevi, and V. Vaikuntanathan. i-hop homomorphic encryption and rerandomizable yao circuits. In *Annual Cryptology Conference*, pages 155–172. Springer, 2010.

[14] S. Gorbunov, V. Vaikuntanathan, and D. Wichs. Leveled fully homomorphic signatures from standard lattices. In *STOC*, pages 469–477. ACM, 2015.

[15] S. D. Gordon, T. Malkin, M. Rosulek, and H. Wee. Multi-party computation of polynomials and branching programs without simultaneous interaction. In T. Johansson and P. Q. Nguyen, editors, *EUROCRYPT*, LNCS. Springer, 2013.

[16] S. Halevi, Y. Ishai, A. Jain, I. Komargodski, A. Sahai, and E. Yogev. Non-interactive multiparty computation without correlated randomness. In T. Takagi and T. Peyrin, editors, *ASIACRYPT*, volume 10626 of *LNCS*, pages 181–211. Springer, 2017.

[17] S. Halevi, Y. Lindell, and B. Pinkas. Secure computation on the web: Computing without simultaneous interaction. In *CRYPTO*, pages 132–150, 2011.

[18] P. Hallgren. *BetterTimes Python Library*, 2017 (accessed January 22, 2020).

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

[20] P. A. Hallgren, M. Ochoa, and A. Sabelfeld. InnerCircle: A parallelizable decentralized privacy-preserving location proximity protocol. In *PST*, pages 1–6, 2015.

[21] P. A. Hallgren, M. Ochoa, and A. Sabelfeld. MaxPace: Speed-Constrained Location Queries. In *CNS*, 2016.

[22] A. Hern. Uber employees 'spied on ex-partners, politicians and Beyoncé', 2016. https://www.theguardian.com/technology/2016/dec/13/uber-employees-spying-ex-partners-politicians-beyonce.

[23] A. Jarrous and B. Pinkas. Canon-mpc, a system for casual non-interactive secure multi-party computation using native client. In A. Sadeghi and S. Foresti, editors, *WPES*, pages 155–166. ACM, 2013.

[24] K. Järvinen, Á. Kiss, T. Schneider, O. Tkachenko, and Z. Yang. Faster privacy-preserving location proximity schemes. In *CANS*, volume 11124 of *LNCS*, pages 3–22. Springer, 2018.

[25] V. Kolesnikov, A.-R. Sadeghi, and T. Schneider. Improved garbled circuit building blocks and applications to auctions and computing minima. In *CANS*, pages 1–20. Springer, 2009.

[26] P. Kotzanikolaou, C. Patsakis, E. Magkos, and M. Korakakis. Lightweight private proximity testing for geospatial social networks. *Computer Communications*, 73:263–270, 2016.

[27] D. Lee. Uber concealed huge data breach, 2017. http://www.bbc.com/news/technology-42075306.

[28] Y. Lindell. How to simulate it–a tutorial on the simulation proof technique. In *Tutorials on the Foundations of Cryptography*, pages 277–346. Springer, 2017.

[29] 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.

[30] A. Narayanan, N. Thiagarajan, M. Lakhani, M. Hamburg, and D. Boneh. Location privacy via private proximity testing. In *NDSS*, 2011.

[31] J. D. Nielsen, J. I. Pagter, and M. B. Stausholm. Location privacy via actively secure private proximity testing. In *PerCom Workshops*, pages 381–386. IEEE CS, 2012.

[32] I. Oleynikov, E. Pagnin, and A. Sabelfeld. Where are you Bob? Privacy-preserving proximity testing with a napping party. 2020.

[33] E. Pagnin, G. Gunnarsson, P. Talebi, C. Orlandi, and A. Sabelfeld. TOP-Pool: Time-aware Optimized Privacy-Preserving Ridesharing. *PoPETs*, 2019(4):93–111, 2019.

[34] 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*, 2015.

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

[36] G. Saldamli, R. Chow, H. Jin, and B. P. Knijnenburg. Private proximity testing with an untrusted server. In *WISEC*, pages 113–118. ACM, 2013.

[37] L. Schabhüser, D. Butin, and J. Buchmann. Context hiding multi-key linearly homomorphic authenticators. In *CT-RSA*, pages 493–513. Springer, 2019.

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

[39] Sharemind MPC Platform. https://sharemind.cyber.ee/sharemind-mpc/multi-party-computation/.

[40] C. Shu. Uber reportedly tracked Lyft drivers using a secret software program named 'Hell'. https://techcrunch.com/2017/04/12/hell-o-uber/, 2017.

[41] L. Siksnys, J. R. Thomsen, S. Saltenis, and M. L. Yiu. Private and flexible proximity detection in mobile social networks. In *MDM*, pages 75–84, 2010.

[42] L. Siksnys, J. R. Thomsen, S. Saltenis, M. L. Yiu, and O. Andersen. A location privacy aware friend locator. In *SSTD*, pages 405–410, 2009.

[43] M. Veytsman. How I was able to track the location of any tinder user, February 2014. http://blog.includesecurity.com/.

[44] G. Zhong, I. Goldberg, and U. Hengartner. Louis, lester and pierre: Three protocols for location privacy. In *PET*, pages 62–76, 2007.

Appendix

1.A Tools Used in OLIC

Homomorphic Encryption [4]. A *Homomorphic Encryption* scheme is a tuple of four PPT algorithms $\mathsf{HE} = (\mathsf{KeyGen}, \mathsf{Enc}, \mathsf{Eval}, \mathsf{Dec})$ that satisfy the properties of correctness, compactness, (semantic) security and circuit privacy. Intuitively these properties states that: given an n-input function f and n ciphertexts $\mathsf{ct}_i = \mathsf{Enc}(pk, m_i)$, the Eval algorithm outputs a new ciphertext ct' that decrypts to $f(m_1, \ldots, m_n)$. The ciphertext ct' is short, and given a ciphertext ct no PPT algorithm can guess what message is encrypted in ct unless given access to the secret key sk for decryption.

Formally, the algorithms are as follows:

$\mathsf{KeyGen}(1^\lambda)$**:** the key generation algorithm takes as input the security parameter λ and outputs a key pair (sk, pk). Implicitly this algorithm also defines the set of plaintext $\mathcal{M}$ and of ciphertexts $\mathcal{C}$.

$\mathsf{Enc}(pk, m)$**:** the encryption algorithm takes as input pk and a message m, and it outputs a ciphertext ct.

$\mathsf{Eval}(pk, f, \mathsf{ct}_1, \ldots, \mathsf{ct}_n)$**:** the evaluation algorithm takes as input pk, a function $f : \mathcal{M}^n \to \mathcal{M}$ in a set of admissible functions $func$ and n ciphertexts. It returns a ciphertext ct.

$\mathsf{Dec}(sk, \mathsf{ct})$**:** the decryption algorithm takes sk and a ciphertext ct, and outputs a message m.

An additive homomorphic encryption scheme is a HE where the set of functions f that Eval can handle is made of linear functions. Concretely this means that given $\mathsf{Enc}(m_1)$ and $\mathsf{Enc}(m_2)$, and two coefficients $a_1, a_2 \in M$, one can efficiently compute $\mathsf{Enc}(a_1 m_1 + a_2 m_2)$.

Function privacy. We adopt the definition of function for honest-but-curious parties given in [4]. In a nutshell, this definition states that the scheme is function-private if there exists an efficient simulator Sim such that for every compatible sequence of admissible functions $f = f_1 \circ \cdots \circ f_t$ the following two distributions are indistinguishable $\mathsf{Eval}_{pk}(f_j, c_{j1}) \stackrel{c}{\equiv} \mathsf{Sim}(pk, c_{j1}, 1^{|f_j|}, (f_1 \circ \cdots \circ f_j)(x))$. For further details we refer the readers to [4], Def. 2.

1.B Detailed Measurements

Table 1.2 reports the concrete running times we obtained in our experiments. These values are used as source to plot the overall running time of each party, in Figure 1.10.

Table 1.2: Running time of parties in InnerCircle and OLIC.

Radius r	Time [s]									
	InnerCircle Alice		InnerCircle Bob		OLIC Alice		OLIC Bob		OLIC servers	
	(EC)	**(non-EC)**	**(EC)**	**(non-EC)**	**(EC)**	**(non-EC)**	**(EC)**	**(non-EC)**	**(EC)**	**(non-EC)**
0	0.01	0.00	0.00	0.00	0.01	0.00	0.10	0.18	0.08	0.04
5	0.03	0.02	0.06	0.04	0.03	0.02	0.10	0.18	0.13	0.08
10	0.10	0.04	0.18	0.14	0.09	0.04	0.10	0.18	0.26	0.18
15	0.17	0.08	0.37	0.28	0.19	0.08	0.10	0.17	0.45	0.32
20	0.36	0.13	0.62	0.48	0.32	0.12	0.10	0.17	0.70	0.51
25	0.47	0.16	0.92	0.71	0.57	0.18	0.10	0.17	1.00	0.75
30	0.61	0.30	1.28	0.99	0.67	0.23	0.10	0.17	1.36	1.03
35	0.77	0.30	1.71	1.31	0.67	0.29	0.10	0.17	1.79	1.35
40	1.16	0.44	2.17	1.67	1.19	0.42	0.10	0.17	2.25	1.71
45	1.47	0.52	2.70	2.08	1.22	0.61	0.10	0.17	2.78	2.12
50	1.63	0.63	3.30	2.55	1.69	0.72	0.10	0.17	3.38	2.59
55	2.23	0.69	3.92	3.04	1.85	0.74	0.10	0.17	4.00	3.08
60	2.29	0.95	4.63	3.59	2.22	0.95	0.10	0.17	4.71	3.63
65	2.53	1.12	5.38	4.19	2.41	1.05	0.10	0.17	5.46	4.22
70	3.19	1.20	6.20	4.84	3.17	1.24	0.10	0.17	6.28	4.88
75	3.33	1.34	7.07	5.52	3.28	1.33	0.10	0.17	7.15	5.56
80	3.82	1.64	7.99	6.26	4.10	1.43	0.10	0.17	8.07	6.30
85	4.87	1.84	8.99	7.07	4.36	1.83	0.10	0.17	9.07	7.10
90	5.07	1.67	10.03	7.91	4.92	1.98	0.10	0.17	10.11	7.95
95	5.79	1.74	11.15	8.79	5.39	1.82	0.10	0.17	11.22	8.83
100	5.93	2.37	12.33	9.75	5.64	2.43	0.10	0.17	12.41	9.79

This page intentionally left blank.

2

CatNap: Leveraging Generic MPC for Actively Secure Privacy-Enhancing Proximity Testing with a Napping Party

Ivan Oleynikov, Elena Pagnin, Andrei Sabelfeld

SECRYPT'2022

Abstract. Proximity testing is at the core of several Location-Based Services (LBS). Despite a series of reported and confirmed abuses, modern LBSs still demand their clients to disclose their locations in plain in order to preform location proximity testing.
This works aims at enhancing proximity testing with privacy. We design CatNap a novel protocol that (1) implements precise Euclidean distance matching; (2) allows matching even if the clients are not online at the same time (the "napping party" feature); (3) is secure against active adversaries (malicious actors that corrupt up to one party); (4) makes black-box use of generic Multi-Party Computation techniques (any future improvement of the underlying building blocks will also boost CatNap); and (5) is efficient: servers run with about 0.03 seconds of CPU time and 5.6MB of communication, while clients perform only a small number of Boolean operations and need just 51 bytes of communication.

This page intentionally left blank.

2.1 Introduction

Location-Based Services (LBS) have gained a steadily increasing role in our lives by providing personalized services based on users' locations, e.g., displaying nearby points of interest, selecting optimal services (e.g., taxi rides), or even triggering specific location-based behaviors (e.g., smart home devices). At the core of most LBS is a *proximity testing (PT)* protocol that allows the system to decide whether some parties lie within a certain proximity of one another. This paper focuses on PT by means of privacy-enhancing protocols and input coordinates (e.g., users know their own locations), which is the main use case for LBS. We acknowledge the existence of other approaches that implement PT via direct communication and measuring signal strength using, e.g., Bluetooth [40] (adopted in some COVID-19 contact tracing apps). While these solutions might provide an accurate distance calculation, they occupy a different niche: in some LBS it might not be possible for users to pick each other signals (e.g., planning for a shared ride between towns; matching with a proximity radius larger than the signal range; or matching with offline users).

Modern taxi services match drivers and passengers according to the proximity of their routes, or the start and endpoints of their journeys. Messaging apps use PT to match users who are in the same area, and online mapping services use it to help users discover close-by places.

In current practice, LBS are full-trust centralized services: to deliver their functionality, they require users to submit their location data to the LBS. This way, the LBS provider knows the location of any active client in their system; and clients cannot check if their data has been used the way they expect, and not misused by the LBS provider or stolen by an attacker who breached the security of LBS. 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 designs a cryptographic protocol that performs proximity testing in a privacy-enhancing way. Such protocol is required to be correct (pro-

vide the right answer) and secure (preserve input privacy) by revealing only the outcome of the PT, and no further information about users' locations. In the remainder of the paper, whenever we refer to PT, we will mean privacy-enhancing proximity testing.

Formalizing PT. There exist multiple approaches to formalizing "location" and "proximity" in PT. The grid-based approach [13, 26, 28, 29, 30, 38, 39, 41] divides the whole plane 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 PT at a cost of a simple equality test, this approach suffers from inherent imprecision. Another alternative is polygon-based matching [9], which becomes less efficient if one wants to approximate a circle with a polygon (but may suit applications like geofencing). We follow the line of work on Euclidean distance-based matching [9, 18, 31, 32], because it is precise and it is natural to some important applications, e.g., messengers, social networks, and taxi. Euclidean distance may serve as an approximation of other measures like Manhattan distance.

In this work, we consider users' locations to be points on a (discretized) Euclidean plane (which can approximate a small enough region of Earth's 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 the 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: CatNap easily generalizes to n-dimensional Euclidean distance-based matching.

Distinguishing Features of CatNap. There are three crucial features that we achieve with CatNap but that were out of reach for previous work [9, 18, 31] on Euclidean distance-based PT:

Offline We adopt the setting of "napping party" [31]: in addition to the two clients who want to use the PT, we introduce two servers that will aid the clients in it. One of the clients can connect to the servers at any moment, submit its location (in a privacy-preserving manner) to them and go offline. The other client will connect to them later, submit its location, wait for the servers to perform matching, and retrieve the result. The clients connect to the servers at possibly disjoint moments of time. In real-life applications, the two servers can be run by independent, mutually distrusting organizations which are providing a single LBS together. Introducing servers is necessary to perform privacy-preserving PT while a client is offline. The use of

two not-colluding servers allows us to remove the requirement for clients to share keys or any other secret information before the protocol starts. As a consequence, the data submitted by a client is not tied to a specific other client and it is up to the servers to decide whom to match the client with.

RadiusInd In [18, 31] the protocol performance depends on R, the proximity radius. This is a significant limitation that makes such protocols practical only for small enough values of R. In contrast, our CatNap's performance (computation, communication, and round complexity) does not depend on the chosen value of R.

ActiveSec From the security viewpoint, for a protocol to be truly practical it needs to be secure against active adversaries (actively secure for short). This means that the protocol preserves its security even if some of the parties get corrupted by the adversary, who maliciously makes them deviate from the protocol specification. As discussed by Oleynikov et al. [31], if the adversary corrupts both servers, it can recover all locations submitted by clients. In this setting, it is impossible to guarantee location privacy and clients' input privacy is lost. We require CatNap to have the best possible active security in the given circumstances: to be secure as long as at least one of the two servers is honest.

The offline feature is particularly distinguishing since most existing PT protocols [9, 18, 19, 31, 35, 41] require the clients (who want to perform the PT of their locations) to communicate directly with one another. This presents a significant limitation to the protocols' applications: in some scenarios, users expect to be matched with their friends or places on the map (e.g. cafes, stores) even when the other clients are not online. Therefore it may be desirable to have an intermediate entity that the clients could interact through. While the use of servers is necessary to perform offline PT, relying on two servers comes with an extra benefit: now the clients can reduce their workload by offloading computations to the servers. Although the servers do not learn the matching outcome, they know which clients requested PT to be run (also how many times and when the users did so); this is a necessary compromise since perfectly hiding the user identities to the servers would introduce an unrealistic performance overhead and negate all the benefits of Offline feature. Concrete server policies for choosing clients to match are very application-specific and are out of the scope of this work. It must be noted that such a policy can be correctly enforced as long as at least one of the two servers honestly follows it; which is realistic in our model, where the protocol security already requires one of the servers to be honest.

Table 3 summarizes the features achieved by our protocol, CatNap, compared to the most relevant recent works. The InnerCircle protocol by Hall-

gren et al. [18] involves two clients who communicate with one another directly, its main drawbacks are passive security and performance proportional to R^2. The protocols $\text{ABY}^{\text{C}}_{\text{Y}}$ and $\text{ABY}^{\text{C}}_{\text{AY}}$ by Järvinen et al. [9] have performance that is independent of the radius value R, but use passively secure two-party computation techniques which implicitly demand clients be online at the same time and interact. The OLIC protocol by Oleynikov et al. [31] is essentially an adaptation of InnerCircle to the two-server setting, and thus it is the first protocol to provide the Offline feature. It inherits some of the drawbacks of InnerCircle [18]: passive security and R^2-dependent performance. These works are further discussed in section 2.5. This paper presents CatNap, the first protocol for privacy-enhancing location PT to achieve all the above three properties.

Table 3: Comparison of CatNap features to the related protocols

Protocol	Offline	RadiusInd	ActiveSec
InnerCircle [18]	–	–	–
$\text{ABY}^{\text{C}}_{\text{Y}}$ and $\text{ABY}^{\text{C}}_{\text{AY}}$ [9]	–	+	–
OLIC [31]	+	–	–
CatNap	+	+	+

Our contribution. This paper presents CatNap, a novel, actively secure protocol for server-aided privacy-enhancing PT. CatNap is the first actively secure PT protocol to achieve practical performance. We provide a formal description of the CatNap protocol and its building blocks. We formally prove its security in Canetti's hybrid model [4], as long as one of the two servers is honest. In addition, we develop a proof of concept implementation of CatNap and compare its performance against InnerCircle [18], OLIC [31], $\text{ABY}^{\text{C}}_{\text{Y}}$ and $\text{ABY}^{\text{C}}_{\text{AY}}$ [9]. Although the InnerCircle, $\text{ABY}^{\text{C}}_{\text{AY}}$, and $\text{ABY}^{\text{C}}_{\text{Y}}$ protocols [9, 18] do not work in the same setting as CatNap (their clients talk directly to one another and are required to be online at the same time), we still include them to see how CatNap compares with server-less PT.

Our evaluations show that CatNap's demands on the servers in terms of amortized computation and communication are quite moderate. For example, performing 2000 matchings requires 0.03 seconds of CPU time (ignoring the network latency) and 6 MB of communication in total per matching. We stress that taking into account only the amortized complexity is practical

since in real-life scenarios LBS providers will be matching large numbers of users and will be able to run a longer precomputation phase. It is worth noting that the improved amortized performance of our protocol comes solely from the MPC techniques edaBits [2], SPDZ2k [1], Tinier [3] which tend to perform better when run multiple times, not the construction we present here. The computation and communication cost for clients is negligible, we ignore it in our benchmarks.

Overview of our technique. We build CatNap using generic *Multi-Party Computation* (MPC) techniques provided out of the box by the MP-SPDZ framework [24]. In our protocol, the clients "outsource" the functionality computation to the servers using the technique of Jakobsen et. al. [7]: each of the two clients secret-shares its location between the servers; the servers input the shares into an MPC protocol, reconstruct them there and evaluate the PT functionality; after that, the servers use a simple masking technique to deliver the result to one of the clients without learning it themselves. Since the PT involves both arithmetic (computing distance between the clients) and non-arithmetic (comparing the distance to the threshold radius R) operations, we combine two MPC protocols: SPDZ2k [1] and Tinier [3], using the former for computation in the arithmetic domain, and the latter for the binary domain. To convert values from arithmetic to binary and vice versa we use the daBits [34] and edaBits [2] techniques.

Assumptions. CatNap is not a fully-featured protocol that can be used for a real-life LBS implementation out of the box, it is best seen as a fundamental building block that can be used by an LBS. CatNap works in the standard setting of MPC protocols [27], the same setting was used for a number of previous PT protocols [9, 18, 31] albeit the (passive) adversary was more limited in those protocols. The assumptions of this model are: parties communicate through secure point-to-point channels (which can be implemented in real life by means of Public Key Infrastructure), at 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. CatNap ensures that the adversary can not do this as long as both servers are not corrupted at the same time.

Scope. The setting of CatNap does not address the data leakage that is allowed by 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. CatNap does not define how

clients specify whom they want to be matched with; such a selection process highly depends on the application, and, as a consequence, should not be implemented by a sub-routine such as CatNap. Also, CatNap does not protect against attacks by a user who might probe the protocol with different maliciously crafted locations trying to learn something about the other users. To mitigate this in a real-life instantiation, it may be necessary to apply some policy similar to MaxPace [19] limiting the queries that a client is allowed to make. Also, CatNap trivially supports replacing two servers with more while still allowing all of them except one to be corrupted. This setting relaxes the security assumption at the cost of extra performance overhead; a similar model with multiple servers is offered by the Sharemind [37] framework.

2.2 Preliminaries

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 11. This functionality is implemented by the edaBits [2] technique. Most of the commands in $\mathcal{F}_{\text{AB-MPC}}$ repeat the functionality on which the edaBits is built, except for the commands $\mathrm{ConvertA2B}$ and $\mathrm{Compare}$ which are implemented using the edaBits technique itself. The $\mathrm{Compare}$ 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 [2]). For the sake of generality, we define $\mathrm{Compare}$ as a standalone command and leave its specification up to specific implementations. The edaBits [2] is implemented in MP-SPDZ [24] framework (which we use for our benchmarks).

Notation We will use the notation $[\![x]\!]_{2^m}$ for value $x \in \mathbb{Z}_{2^m}$ being input into the $\mathcal{F}_{\text{AB-MPC}}$ with type = $\mathrm{arithmetic}$, and $[\![x]\!]_2$ for value $x \in \{0,1\}$ with type = binary (the variable names x are assumed to be unique over both $\mathrm{arithmetic}$ 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., $[\![x]\!]_{2^m} \leftarrow \mathrm{ConvertB2A}([\![y]\!]_2)$ means sending $(\mathrm{ConvertB2A}, \text{“}x\text{”}, \text{“}y\text{”})$ to the $\mathcal{F}_{\text{AB-MPC}}$. We will also use values $[\![\cdot]\!]_{2^m}$ 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 $\mathrm{LinComb}$. 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 $\mathrm{IP}(u,v) = \sum_{i=0}^{k-1} u_i v_i = \bigoplus_{i=0}^{k-1} u_i \wedge v_i$. 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 $[\![\overrightarrow{v}]\!]_2 = ([\![v_0]\!]_2, \dots [\![v_{k-1}]\!]_2)$,

Input: On input $(\text{Input}, P_i, \text{type}, \text{id}, x)$ from P_i and $(\text{Input}, P_i, \text{type}, \text{id})$ from all other parties, with id a fresh identifier, type $\in \{\text{binary}, \text{arithmetic}\}$ and $x \in \mathbb{Z}_2$ or $x \in \mathbb{Z}_{2^k}$ (depending on type), store $(\text{type}, \text{id}, x)$.

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

Multiply: On input $(\text{Mult}, \text{type}, \text{id}, \text{id}_1, \text{id}_2)$ 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 2^m if type = arithmetic, and store (id, z).

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

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

Compare: On input $(\text{Compare}, \text{id}, \text{id}', y)$ from all parties, where $y \in \mathbb{Z}_{2^m}$, retrieve $(\text{arithmetic}, \text{id}, x)$, store $(\text{binary}, \text{id}', 1)$ if $x \leq y$ or $(\text{binary}, \text{id}', 0)$ otherwise.

Output: On input $(\text{Output}, \text{type}, \text{id})$ 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 11: Ideal functionality $\mathcal{F}_{\text{AB-MPC}}$ of MPC arithmetic blackbox modulo 2 and modulo 2^k [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

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

2.3 The CatNap Protocol

The CatNap protocol is built by combining the previous works in a blackbox way, i.e., relying only on their most standard properties. Figure 12 gives an

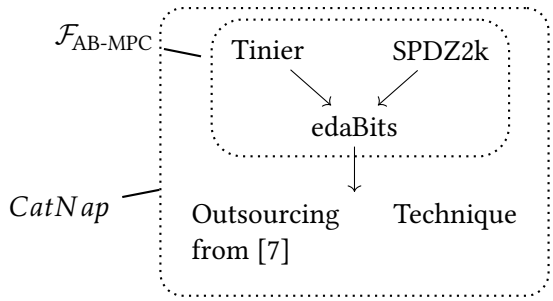


Figure 12: The diagram of blackbox applications of previous works that yields CatNap protocol and the $\mathcal{F}_{\text{AB-MPC}}$ functionality that we use to build CatNap.

overview of the order in which the existing techniques are applied to one another. Here, Tinier provides MPC computations in the binary domain, SPDZ2k provides computations in the 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 transfer their data into the edaBits MPC and then get back the result even if one of the servers is untrusted. The rest of this section shows the operations done by CatNap in greater detail; it essentially unfolds the last step from Figure 12 to show how the inputs and outputs are transferred to and from edaBits MPC, and it also shows how the squared distance between parties is computed and compared to the radius. The edaBits is still treated as a blackbox in this section, since unfolding that one as well would yield too much detail and harm the high-level exposition.

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 $(x_a, y_a) \in \mathbb{Z}_{2^k}^2$ and $(x_b, y_b) \in \mathbb{Z}_{2^k}^2$ 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 13: The $\mathcal{F}_{\text{PT}}$ ideal functionality

CatNap involves four parties: two servers Server-1 and Server-2; and two clients Alice and Bob. Alice and Bob know their respective locations

(x_a, y_a) and (x_b, y_b), and will input these at the start of the protocol. At the end of its execution, CatNap returns to Alice a bit ρ; $\rho = 1$ if her distance to Bob is less than or equal to a given public value R, otherwise $\rho = 0$. Following the offline feature introduced by OLIC, in CatNap clients never exchange messages with one another directly: all of Bob's interaction happens before any interaction from Alice (i.e., Bob acts as a "napping party" during the actual proximity test). Figure 13 shows the formal definition of the ideal functionality $\mathcal{F}_{\text{PT}}$ that CatNap implements, while Figure 15 shows how CatNap implements $\mathcal{F}_{\text{PT}}$ in the real world.

CatNap achieves the $\mathcal{F}_{\text{PT}}$ functionality in three major steps. First, the client inputs are transferred into the $\mathcal{F}_{\text{AB-MPC}}$ functionality. Remember that clients cannot communicate with the $\mathcal{F}_{\text{AB-MPC}}$ directly, only servers do that. To transfer its input, each client authenticates it using AMD (Algebraic Manipulation Detection code) and secret-shares z, its authentication key, and tag between the servers [7]. The servers input the shares together with the authentication tags into $\mathcal{F}_{\text{AB-MPC}}$, verify that the shares are correct, and reconstruct them inside the functionality. Second, the servers compute the squared Euclidean distance between the clients' input locations:

$$D = (x_a - x_b)^2 - (y_a - y_b)^2. \tag{2.1}$$

Subsequently, the servers compare D to R^2, obtaining a single bit $\rho \in \{0, 1\}$, where $\rho = 1$ if $D \leq R^2$, and $\rho = 0$ otherwise. We remark that all computations performed by the servers so far are implemented trivially using the arithmetic and comparison operations supported by $\mathcal{F}_{\text{AB-MPC}}$. This means that the servers never see D or the client inputs in plain, yet by interacting with the $\mathcal{F}_{\text{AB-MPC}}$ functionality they can operate on these values without seeing them. Third, the servers transfer the result ρ to one of the clients in a safe way. This is achieved via the technique of Jakobsen et al. suggest in [7]. None of the three steps reveals anything about the clients' inputs or ρ to the servers; all of the values the servers work with are either blinded with random masks or are inside $\mathcal{F}_{\text{AB-MPC}}$.

The transferring of client inputs into the $\mathcal{F}_{\text{AB-MPC}}$ functionality mentioned above is done in bit-decomposed form: each coordinate is represented as a bit-vector, the vectors of all coordinates are concatenated and the result is transferred (using Transfer shown below) into $\mathcal{F}_{\text{AB-MPC}}$. This has a useful side-effect: we can naturally bound inputs of each client by limiting the number of bits in their representation (Transfer accepts only fixed number of bits). This way, a malicious client cannot input values that are too large and cause an overflow modulo 2^m in the computation of D (Equation (2.1)). We limit each client coordinate to k bits, where k is any positive integer such

that $2k + 3 \leq m$ (this ensures that there is no overflow in the expression for D from Equation (2.1)). I.e. for all meaningful values of R, it must hold that $0 \leq R \leq 2^k\sqrt{2}$.

The Transfer **Sub-protocol.** Figure 14 shows the sub-protocol that transfers the clients' inputs into $\mathcal{F}_{\text{AB-MPC}}$. This happens between a client (who can be either Alice or Bob) and the two servers. The purpose of this sub-protocol is to transfer a vector $z \in \mathbb{F}_2^l$ from the client into the binary domain of the $\mathcal{F}_{\text{AB-MPC}}$ functionality (without revealing it to the servers). Formally, this protocol can work for values z of any length. In practice, each client will execute this sub-protocol exactly once with z being the concatenation of the bit-decomposition of their input locations (Alice will additionally concatenate a random bit ρ to her z, which will be used in the last step of the whole CatNap protocol. More on this on Figure 15).

The Transfer routine starts with a client, say, Alice authenticating her input z using AMD with a freshly generated key (step 2), then she secret-shares the value z, the picked key and the authentication tag between the two servers using XOR (steps 1 and 3). The servers input the shares and tags into $\mathcal{F}_{\text{AB-MPC}}$ (step 4). At this point, the servers can simply reveal the keys to each other (steps 5), since they cannot modify the shares nor the tags they input into the functionality. After that, the servers recompute the authentication tag $[\![\overrightarrow{u}]\!]_2$ (step 6) and compare it to the one that the servers have input (step 7).

Computing AMD is essentially free since it uses only linear operations (as shown in Appendix 2.A). On the other hand, the equality check is the heaviest step computations-wise, because this comparison requires non-linear Boolean operation $\bigvee$. The servers reveal the result $[\![c]\!]_2$ of the equality check and abort if $[\![c]\!]_2 = 0$ (step 8). This completes the authentication check, now each server is convinced that the other one has not cheated while inputting the client data into $\mathcal{F}_{\text{AB-MPC}}$. Now, they can reconstruct the secret-shared value $[\![\overrightarrow{z}]\!]_2$ (without revealing it yet), which is the result of running this sub-protocol.

The CatNap Protocol Figure 15 provides a detailed overview of our CatNap protocol. We recall that CatNap implements the $\mathcal{F}_{\text{PT}}$ functionality from Figure 13. The protocol starts with both clients transferring their inputs into $\mathcal{F}_{\text{AB-MPC}}$ using Transfer (step 1). They do so by running the Transfer protocol on the concatenation of the bit-decomposition of their inputs. Alice additionally transfers a random bit μ that will be used in the final stage of CatNap to privately transfer the matching outcome ρ from $\mathcal{F}_{\text{AB-MPC}}$ back to her. The servers convert the clients' inputs from the binary domain into the arithmetic domain, as required in the $\mathcal{F}_{\text{AB-MPC}}$ functionality (step 1c). Alice's

mask μ remains in the binary domain.

Once the clients' inputs are in $\mathcal{F}_{\text{AB-MPC}}$ and ready to be used, the servers can compute the squared distance and compare it to R^2 (step 2). All this is trivially done using commands supported by $\mathcal{F}_{\text{AB-MPC}}$. The outcome of this comparison, ρ, which is also the result of matching, is stored in $[\![\rho]\!]_2$ inside $\mathcal{F}_{\text{AB-MPC}}$. At this point, the only thing that needs to be done is revealing the result $[\![\rho]\!]_2$ to Alice (without leaking anything to anyone else). To achieve this, we mask it with Alice's random bit μ and open the masked value ρ' (step 3) to both servers. Since the value is masked, the servers cannot learn anything about it. Moreover, since both servers hold a copy of the masked result, none of them can modify it without getting caught. Both servers forward ρ' to Alice (step 4), who makes sure that both servers sent the same value, and unmasks it to obtain the matching result ρ (step 5).

2.3.1 Security Proof

To prove the security of CatNap (Figure 15) we must show that it securely implements [27] the functionality $\mathcal{F}_{\text{PT}}$ (Figure 11) in the presence of static active adversary who can corrupt any subset of parties as long as one of the servers is not corrupted.

Since the protocol CatNap is modular and is built from off-the-shelf MPC techniques (shown on Figure 12), we prove its security by combining the proofs of the corresponding techniques. Showing here all the details would be too technical and not very insightful, 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 [1, 2, 3, 7] and check the details). The main objective of this section is to show that the techniques from Figure 12 fit each other.

Combining of edaBits with Tinier and SPDZ2k 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.

Parameters: a positive number R, the radius of proximity testing; k, the bit width of client coordinates.
Setup: Alice, Bob and the two servers. The servers have access to the $\mathcal{F}_{\text{AB-MPC}}$ functionality (depicted in Figure 11). It must hold that $2k + 3 \leq m$, where Z_{2^m} is the arithmetic domain of $\mathcal{F}_{\text{AB-MPC}}$. Alice and Bob receive (x_a, y_a) and (x_b, y_b) as inputs.

1. Inputs outsourcing phase.
   (a) Bob bit-decomposes his input coordinates x_b and y_b, represents them as a single $2k$ bit string, and runs the Transfer protocol (Figure 14) on it.
   (b) Alice samples a random bit μ, bit decomposes her inputs x_a and y_a, and represents all of them as a single string of $2k + 1$ bits. Then she runs the Transfer protocol on it.
   (c) The servers convert the client inputs into the arithmetic domain
   $$[\![x_a]\!]_{2^m} \leftarrow \text{ConvertB2A}([\![\overrightarrow{x_a}]\!]_2)$$
   $$[\![y_a]\!]_{2^m} \leftarrow \text{ConvertB2A}([\![\overrightarrow{y_a}]\!]_2)$$
   $$[\![x_b]\!]_{2^m} \leftarrow \text{ConvertB2A}([\![\overrightarrow{x_b}]\!]_2)$$
   $$[\![y_b]\!]_{2^m} \leftarrow \text{ConvertB2A}([\![\overrightarrow{y_b}]\!]_2).$$
   The value $[\![\mu]\!]_2$ stays in the binary domain.
2. The Servers compute the squared distance between Alice and Bob and compare it to R^2:
   (a) $[\![D]\!]_{2^m} \leftarrow ([\![x_a]\!]_{2^m} - [\![x_b]\!]_{2^m})^2 + ([\![y_a]\!]_{2^m} - [\![y_b]\!]_{2^m})^2$
   (b) $[\![\rho]\!]_2 \leftarrow \text{Compare}([\![D]\!]_{2^m}, R^2)$.
3. The servers mask the bit ρ with μ and reveal the result:
   (a) $[\![\rho']\!]_2 \leftarrow [\![\rho]\!]_2 \oplus [\![\mu]\!]_2$
   (b) $\rho' \leftarrow \text{Output}([\![\rho']\!]_2)$.
4. Both servers forward the obtained ρ' to Alice.
5. Alice ensures that both servers have sent the same value of ρ', unmasks it to get the final result $\rho = \rho' \oplus \mu$, which she outputs.

Figure 15: The CatNap protocol

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 11) combines both. This issue is resolved by noting that outsourcing of computation does not restrict the operations that the employed MPC allows are supported (as long as field addition and multiplication are available). This allows us to present edaBits to the outsourcing technique as if it was only binary MPC; input and output

operations that CatNap performs (Figures 14 and 15) on $\mathcal{F}_{\text{AB-MPC}}$ work with bits while all the arithmetic operations are done only inside $\mathcal{F}_{\text{AB-MPC}}$.

2.4 Evaluation

To evaluate the performance of CatNap, we implemented it in the MP-SPDZ [24] cryptographic framework and made it available online [30]. We compare its performance to InnerCircle, $\text{ABY}^{\text{C}}_{\text{AY}}$ and $\text{ABY}^{\text{C}}_{\text{Y}}$, OLIC. Because of the inherent similarity between InnerCircle and OLIC, we run only OLIC in our benchmarks and argue that most of the conclusions we make here about OLIC hold for InnerCircle as well. For the performance comparison, we focus total execution time (on a single CPU core) and on total data exchanged by parties.

To achieve a fairer comparison, we ran all the protocols on the same Linux machine having Intel(R) Core(TM) i7-8700 CPU and 32 GB of RAM. For each of the protocols we run here we use the implementation provided by their original papers: 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), such comparison is useful nevertheless. We do not introduce any intentional network latency, 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.

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.

$\text{ABY}^{\text{C}}_{\text{AY}}$ and $\text{ABY}^{\text{C}}_{\text{Y}}$. We use ABY [10] parameters of the original paper [9]: `bits` = 64, `secparam` = 128. In other words, the values domain is 2^{64} and the symmetric security key length is 128 bits.

CatNap. We instantiate DPDZ2k and Tinier with the security parameter of 64 bits, and plaintext values of SPDZ2k consist of 64 bits. The statistical security parameter for edaBits is 40.

We do not include the performance of clients in our benchmarks of CatNap since it is negligible; as can be seen on Figures 15 and 14, the total communication cost for each client does not exceed $2(3\sigma + 4k + 1)$ bits (which

is 51 bytes for $k = 20$ and $\sigma = 40$), and the computation cost constitutes a small number of Boolean operations.

Figure 16 shows the amortized performance of server in CatNap depending on the number of times the protocol is executed. These measurements include both setup time and the actual protocol execution. As the number of repetitions approaches 4000, the amortized execution time reaches 0.03 seconds, and the total communication cost reaches 5.6 MB. We use these two numbers as constants in the next plots, where we compare CatNap to other protocols. The B parameter present on the plots is internal to the edaBits; smaller values of B are expected to provide better asymptotic performance.

The performance of OLIC depends on the specific value used for the radius R, this is reflected in the measurements presented on Figure 17. The protocols that have performance independent of R are shown there as straight horizontal lines. Notably, CatNap is less efficient than $\mathrm{ABY}^{\mathrm{C}}_{\mathrm{AY}}$ and $\mathrm{ABY}^{\mathrm{C}}_{\mathrm{Y}}$ (we consider it a minor price to pay since CatNap achieves active security), but it still becomes more efficient than OLIC for large enough values of R.

2.5 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 PT, and Bob must be present online to perform the PT. The Lester protocol does not use a third party but rather than performing PT 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 PT. They cast the PT problem as equality testing on a grid system of hexagons. One of the proposed protocols utilizes an oblivious server. Parties in this protocol use symmetric encryption, which leads to better performance. However, this requires having preshared keys among parties, which is less amenable to one-to-many PT. 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 PT, using additively homomorphic encryption between two parties that must be online. 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 PT using elliptic curves. They require Alice and Bob to be online to be able to run the protocol. Järvinen et al. [9] 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 the requirement of the two parties being online applies to their setting as well. Hallgren et al. [17] show how to leverage PT for endpoint-based ridesharing, building on the InnerCircle protocol, 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. Some of these protocols are mentioned in Table 3.

The main challenge of Euclidean distance-based PT 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. To overcome this, we use state-of-the-art MPC techniques: SPDZ2k protocol for arithmetic computation [1], Tinier [3] for Boolean computation and edaBits [2] for converting values between Boolean and arithmetic domains.

In the wake of the COVID-19 pandemic, privacy-preserving PT witness a boom of protocols that rely on Bluetooth communication [40]. These solutions realize PT 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 rely on a specific technology (e.g., Bluetooth communication) and aims at providing precise matching using the Euclidean distance. 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 [9, 13, 18, 19, 29, 31, 35, 36, 37, 38, 39, 41] of the existing approaches to proximity testings require both parties to be online or requires clients to share common keys before the protocol starts, thus not being suitable for one-to-many matching, and also provide only passive security, limiting the practical applicability of the protocol. A notable exception to the work above is the C-Hide&Hash protocol by Mascetti et al. [28], which allows one-to-many testing, yet at the price of not computing the precise proximity result but its grid-based approximation. Generally, a large number of approaches [13, 26, 28, 29, 30, 38, 39, 41] resort to grid-based approximations, thus losing precision of proximity tests.

2.6 Conclusion

We presented CatNap, a secure and privacy-enhancing protocol for PT, which performs exact Euclidean distance-based matching. CatNap solves some of the major issues previous similar works suffered from: its performance does not depend on the proximity radius; it is secure against active adversaries; and it does not require clients to be simultaneously online for the PT to run. Our evaluation results confirm that the amortized performance of CatNap is practical: the running time per repetition is close to negligible, and the communication cost is around a few megabytes.

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 taxi applications (e.g. BlaBlaCar [3]), where drivers need to pick up the passengers at the right time (and get the actual passenger location if the matching succeeded). We also allow one-to-many matching via the "napping party" feature, since the servers can reuse Alice and Bob's locations multiple times. For example, Bob can submit his location to the servers and let them match him with any of his friends, yielding a single bit of the result or a list of all of his friends who are nearby. In the case of one-to-many matching, the overhead of our approach will grow linearly in the number of clients for the servers and stay constant for the clients. Also, since the protocol already relies on one of the servers being honest, this fact can be used to implement a fine-grained policy to control whom a certain client can be matched with, track the exact time when the client has submitted their location to the servers (to show the other clients how fresh it is), or let the client see who requested matching with them while they were offline; these features are orthogonal to our work and are dependent on a specific application scenario.

CatNap 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 some 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 breaking it harder.

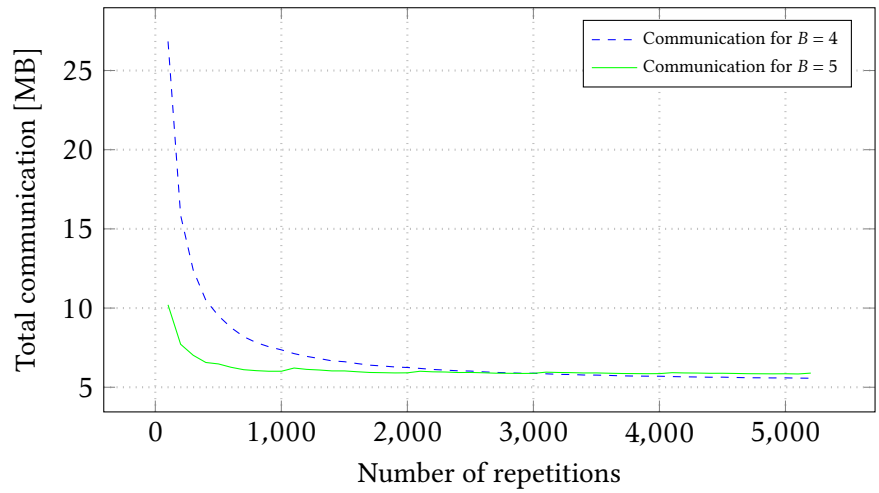
We leave a more extensive evaluation of CatNap's performance in the presence of realistic network latency for the future work, as well as the evaluation of time-based matching. Other possible directions of future work include building protocols for server-less PT, which would be based on black-box use of generic MPC; and improving the efficiency of the MAC construction based on Toeplitz matrix that we use in CatNap.

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).

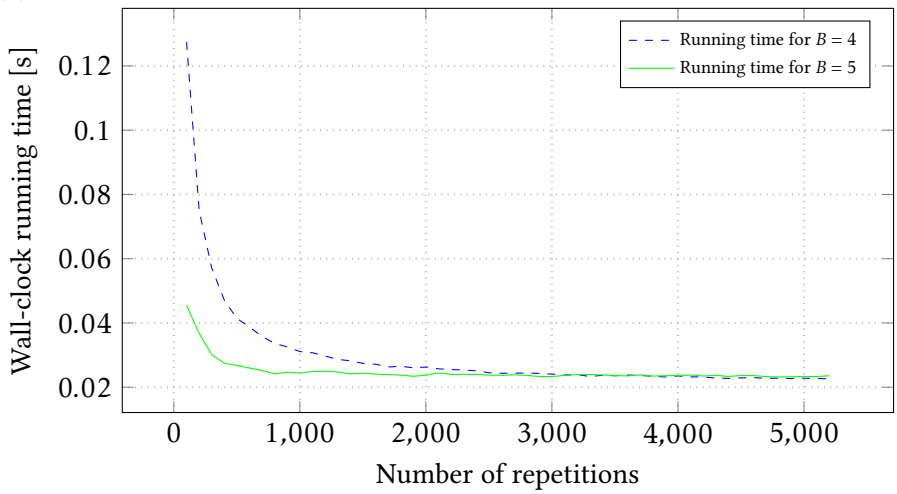
Setup: One client (Alice or Bob) and the two servers (Server-1 and Server-2). The servers have access to the $\mathcal{F}_{\text{AB-MPC}}$ functionality (of Figure 11).
Initial condition: The client knows its input $z \in \mathbb{F}_2^l$, which is a sequence of l bits. σ is a statistical security parameter.
Final condition: The bits of z are input into $\mathcal{F}_{\text{AB-MPC}}$ functionality as $[\![\overrightarrow{z}]\!]_2$.

1. The client authenticates its input z using AMD with freshly chosen key:
   (a) $\kappa \leftarrow^{\$} \mathbb{F}_{2^\sigma}$
   (b) $t = \text{AMD}_\kappa(z)$
2. The client secret-shares its input z, the authentication key and tag input z using AMD with freshly chosen key:
   (a) $r^{(1)} \leftarrow^{\$} \mathbb{F}_{2^l}$, $\kappa^{(1)} \leftarrow^{\$} \mathbb{F}_{2^\sigma}$
   (b) $r^{(2)} = z \oplus r^{(1)}$
   (c) $\kappa^{(2)} = \kappa \oplus \kappa^{(1)}$
   (d) $t^{(1)} \leftarrow^{\$} \mathbb{F}_{2^\sigma}$
   (e) $t^{(2)} = t \oplus t^{(1)}$
3. The client sends $(r^{(1)}, \kappa^{(1)}, t^{(1)})$ to Server-1, and $(r^{(2)}, \kappa^{(2)}, t^{(2)})$ to Server-2.
4. The servers input shares $r^{(\cdot)}$ and the tags $t^{(\cdot)}$ into the $\mathcal{F}_{\text{AB-MPC}}$
   (a) $[\![r_i^{(1)}]\!]_2 \leftarrow \text{Input}_{\text{Server-1}}(r_i^{(1)})$ for $i \in \{0 \dots l-1\}$
   (b) $[\![r_i^{(2)}]\!]_2 \leftarrow \text{Input}_{\text{Server-2}}(r_i^{(2)})$ for $i \in \{0 \dots l-1\}$
   (c) $[\![t_i^{(1)}]\!]_2 \leftarrow \text{Input}_{\text{Server-1}}(t_i^{(1)})$ for $i \in \{0 \dots \sigma-1\}$
   (d) $[\![t_i^{(2)}]\!]_2 \leftarrow \text{Input}_{\text{Server-2}}(t_i^{(2)})$ for $i \in \{0 \dots \sigma-1\}$.
5. The servers send $\kappa^{(1)}$ and $\kappa^{(2)}$ to one another and recover $\kappa = \kappa^{(1)} \oplus \kappa^{(2)}$.
6. The servers recompute the tag for the z inside the $\mathcal{F}_{\text{AB-MPC}}$:
   (a) $[\![\overrightarrow{u}]\!]_2 = \text{AMD}_\kappa([\![\overrightarrow{r^{(1)}}]\!]_2 \oplus [\![\overrightarrow{r^{(2)}}]\!]_2)$
7. The servers check that the computed tags match the expected values:
   $$[\![c]\!]_2 \leftarrow \text{EQ}([\![\overrightarrow{u}]\!]_2, [\![\overrightarrow{t^{(1)}}]\!]_2 \oplus [\![\overrightarrow{t^{(2)}}]\!]_2),$$
   where $\text{EQ}((a_0 \dots a_{l-1}), (b_0 \dots b_{l-1})) = \neg \bigvee_{i=0}^{l-1} a_i \oplus b_i$ is the logical formula that compares two sequences of bits for equality.
8. The servers reveal the bit $c \leftarrow \text{Output}([\![c]\!]_2)$ and abort if $c = 0$.
9. The servers reconstruct the value $[\![\overrightarrow{z}]\!]_2 = [\![\overrightarrow{r^{(1)}}]\!]_2 \oplus [\![\overrightarrow{r^{(2)}}]\!]_2$, which is the result of this sub-protocol.

Figure 14: The Transfer sub-protocol



(a) Total communication



(b) Running time

Figure 16: Amortized performance of CatNap by the number of repetitions

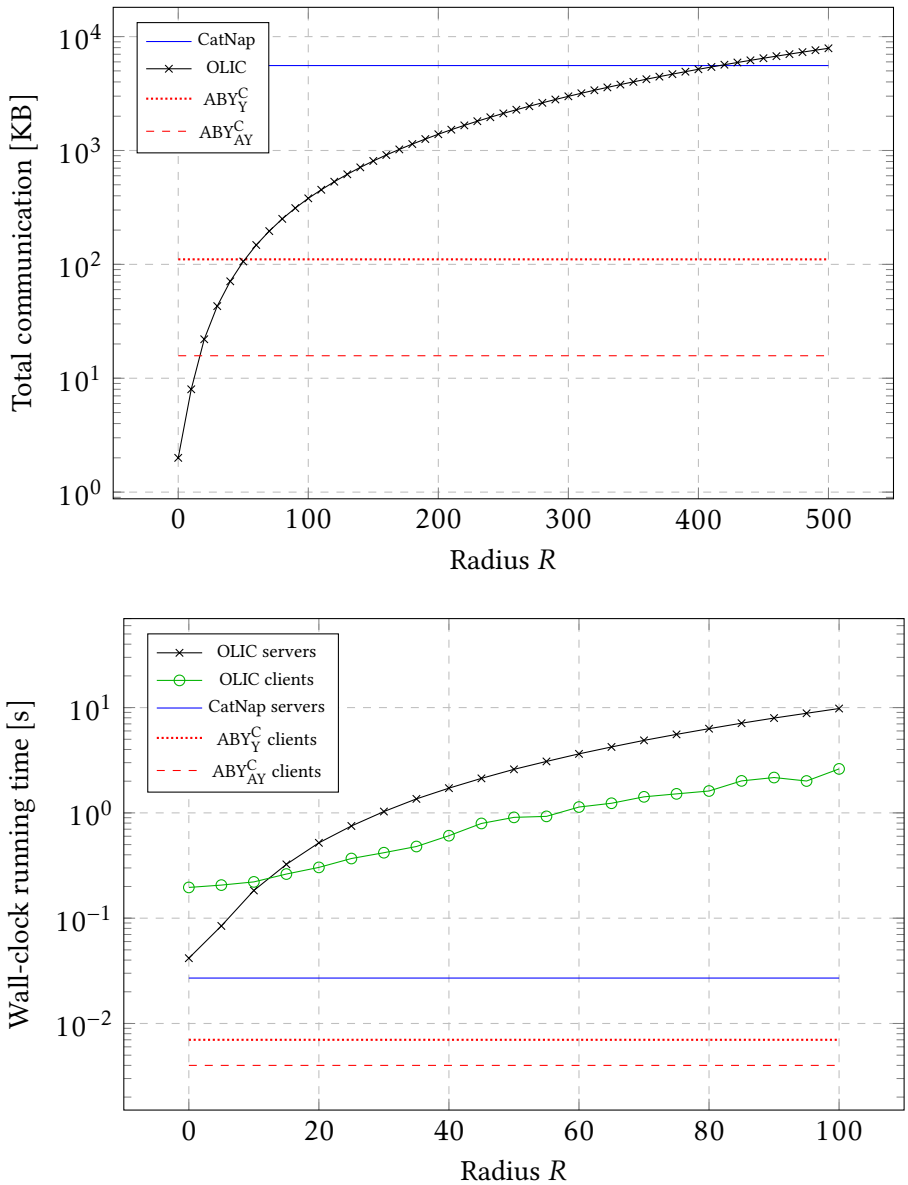


Figure 17: Comparison of CatNap with OLIC, ABY$^{C}_{Y}$ and ABY$^{C}_{AY}$

Bibliography

[1] A. Beimel, A. Gabizon, Y. Ishai, E. Kushilevitz, S. Meldgaard, and A. Paskin-Cherniavsky. Non-interactive secure multiparty computation. In *CRYPTO*, LNCS, 2014.

[2] F. Benhamouda, H. Krawczyk, and T. Rabin. Robust non-interactive multiparty computation against constant-size collusion. In *CRYPTO*, LNCS, 2017.

[3] BlaBlaCar - Trusted carpooling. https://www.blablacar.com/, 2022.

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

[5] S. 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 22-Mar-2022].

[6] J. 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 22-Mar-2022].

[7] J. Cox and M. 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 22-Mar-2022].

[8] R. Cramer, I. Damgard, D. Escudero, P. Scholl, and C. Xing. Spdz2k: Efficient mpc mod 2^k for dishonest majority. In *CRYPTO*, 2018.

[9] D. Demmler, T. Schneider, and M. Zohner. ABY - A framework for efficient mixed-protocol secure two-party computation. In *NDSS*, 2015.

[10] D. Escudero, S. Ghosh, M. Keller, R. Rachuri, and P. Scholl. Improved primitives for mpc over mixed arithmetic-binary circuits. In *CRYPTO*, 2020.

[11] T. K. Frederiksen, M. Keller, E. Orsini, and P. Scholl. A unified approach to mpc with preprocessing using ot. In *ASIACRYPT*, 2015.

[12] 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.

[13] S. D. Gordon, T. Malkin, M. Rosulek, and H. Wee. Multi-party computation of polynomials and branching programs without simultaneous interaction. In *EUROCRYPT*, LNCS. Springer, 2013.

[14] S. Halevi, Y. Ishai, A. Jain, I. Komargodski, A. Sahai, and E. Yogev. Non-interactive multiparty computation without correlated randomness. In *ASIACRYPT*, LNCS, 2017.

[15] S. Halevi, Y. Lindell, and B. Pinkas. Secure computation on the web: Computing without simultaneous interaction. In *CRYPTO*, 2011.

[16] P. Hallgren, C. Orlandi, and A. Sabelfeld. PrivatePool: Privacy-Preserving Ridesharing. In *CSF*, Aug 2017.

[17] P. A. Hallgren, M. Ochoa, and A. Sabelfeld. InnerCircle: A parallelizable decentralized privacy-preserving location proximity protocol. In *PST*, 2015.

[18] P. A. Hallgren, M. Ochoa, and A. Sabelfeld. MaxPace: Speed-Constrained Location Queries. In *CNS*, 2016.

[19] A. 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 22-Mar-2022].

[20] T. P. Jakobsen, J. B. Nielsen, and C. Orlandi. A framework for outsourcing of secure computation. In *ACM CCSW*, 2014.

[21] A. Jarrous and B. Pinkas. Canon-mpc, a system for casual non-interactive secure multi-party computation using native client. In *WPES*. ACM, 2013.

[22] K. Järvinen, A. Kiss, T. Schneider, O. Tkachenko, and Z. Yang. Faster privacy-preserving location proximity schemes for circles and polygons. *IET Information Security*, 14, 10 2019.

[23] M. Keller. Mp-spdz: A versatile framework for multi-party computation. In *ACM SIGSAC*, 2020.

[24] P. Kotzanikolaou, C. Patsakis, E. Magkos, and M. Korakakis. Lightweight private proximity testing for geospatial social networks. *Computer Communications*, 2016.

[25] Y. Lindell. How to simulate it - a tutorial on the simulation proof technique. ePrint, 2016. https://eprint.iacr.org/2016/046.

[26] 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.*, 2011.

[27] A. Narayanan, N. Thiagarajan, M. Lakhani, M. Hamburg, and D. Boneh. Location privacy via private proximity testing. In *NDSS*, 2011.

[28] J. D. Nielsen, J. I. Pagter, and M. B. Stausholm. Location privacy via actively secure private proximity testing. In *PerCom Workshops*. IEEE CS, 2012.

[29] I. Oleynikov, E. Pagnin, and A. Sabelfeld. Where are you Bob? Privacy-Preserving Proximity Testing with a Napping Party. In *ESORICS*, 2020.

[30] I. Oleynikov, E. Pagnin, and A. Sabelfeld. Catnap: Leveraging generic mpc for actively secure privacy-enhancing proximity testing with a napping party (extended version), 2022. https://www.cse.chalmers.se/research/group/security/catnap/.

[31] E. Pagnin, G. Gunnarsson, P. Talebi, C. Orlandi, and A. Sabelfeld. TOPPool: Time-aware Optimized Privacy-Preserving Ridesharing. *PoPETs*, 2019.

[32] 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*, 2015.

[33] D. Rotaru and T. Wood. Marbled circuits: Mixing arithmetic and boolean circuits with active security. In *INDOCRYPT 2019*, 2019.

[34] M. N. Sakib and C. Huang. Privacy preserving proximity testing using elliptic curves. In *ITNAC*, 2016.

[35] G. Saldamli, R. Chow, H. Jin, and B. P. Knijnenburg. Private proximity testing with an untrusted server. In *WISEC*. ACM, 2013.

[36] J. Sedenka and P. Gasti. Privacy-preserving distance computation and proximity testing on earth, done right. In *AsiaCCS*, 2014.

[37] Sharemind MPC Platform. https://sharemind.cyber.ee/sharemind-mpc/multi-party-computation/, 2022.

[38] L. Siksnys, J. R. Thomsen, S. Saltenis, and M. L. Yiu. Private and flexible proximity detection in mobile social networks. In *MDM*, 2010.

[39] L. Siksnys, J. R. Thomsen, S. Saltenis, M. L. Yiu, and O. Andersen. A location privacy aware friend locator. In *SSTD*, 2009.

[40] C. Troncoso, M. Payer, J.-P. Hubaux, M. Salathé, J. Larus, E. Bugnion, W. Lueks, T. Stadler, A. Pyrgelis, D. Antonioli, et al. Decentralized privacy-preserving proximity tracing. *arXiv preprint arXiv:2005.12273*, 2020.

[41] G. Zhong, I. Goldberg, and U. Hengartner. Louis, lester and pierre: Three protocols for location privacy. In *PET*, 2007.

Appendix

2.A Algebraic Manipulation Detection Code

In this section, we define the Algebraic Manipulation Detection (AMD) code employed in the outsourcing technique [7] used in CatNap to transfer the client inputs into the functionality $\mathcal{F}_{\text{AB-MPC}}$.

The AMD code allows one to authenticate some data with a private key in way that is similar to Message Authentication Code (MAC), but with a different security property. In the case of MAC, the adversary (who doesn't know the key) is allowed to obtain a number of messages with their corresponding authentication tags; and the adversary's task is to forge the authentication tag for some new message of its choice. In case of AMD codes, the adversary must forge the authentication of a message by introducing additive error into each of message, key and tag; the following definition states this property formally.

Definition 1 ([7]). *The function $f_k(x_1, \dots x_l) = t$, where $k, x_i, t \in \mathbb{F}$ for some finite field $\mathbb{F}$, is called an Algebraic Manipulation Detection code if for any $(x_1, \dots x_l)$,ε_t, ε_k, ε_{x_i} the following holds with negligible probability*

$$f_k(x) + \varepsilon_t = f_{k+\varepsilon_k}(x_1 + \varepsilon_{x_1}, x_2 + \varepsilon_{x_2}, \dots, x_l + \varepsilon_{x_l}),$$

for k chosen uniformly at random.

In the definition above, the values ε_k, ε_t, and ε_{x_i} are the errors chosen by the adversary and introduced to the equation $t = f_k(x_1, \dots x_l)$. The adversary breaks the AMD only if the equation still holds after introducing the errors.

Together with the definition we replicate above, Jakobsen et. al. [7] have also suggested an efficient construction of an AMD code:

$$f_k(x_1, x_2 \dots x_l) = k^{l+2} + \sum_{i=1}^{l} x_i k^i.$$

The f function satisfies the Definition 1, where the $\log|\mathbb{F}|$ acts as the security parameter (the adversary's winning probability is bounded using this value).

The values that we authenticate using f in CatNap are represented as bits, therefore we need to map bit vectors into some large enough field $\mathbb{F}$ to authenticate them. Here, we also use a trick from Jakobsen et. al.: take $\mathbb{F}$ to be $\mathbb{F}_{2^\sigma} = \mathbb{F}_2[x]/(p(x))$, i.e., the field of polynomials with coefficients from $\mathbb{F}_2 = \{0,1\}$ where all the field operations are performed modulo an irreducible polynomial $p(x)$ with $\deg p = \sigma$. Any vector $v = (v_0, \ldots, v_{\sigma-1})$ is now mapped to the polynomial $q_v(x) = \sum_{i=0}^{\sigma-1} v_i x^i$, the addition of the polynomials is simple component-wise XOR of the corresponding vectors, and the multiplication is the usual polynomial product reduced modulo $p(x)$.

The key feature of this mapping is that it preserves linearity. The function $(q_v \mapsto q_u q_v)$ can be computed using only XORs of the q_v's coefficients. It means that we can compute such funcitons inside $\mathcal{F}_{\text{AB-MPC}}$ (Figure 11) virtually for free, and the performance overhead of AMD validation is going to be low.

Next, we define the complete AMD code that can handle bit vectors of arbitrary length by the function $\text{AMD}_k(b_1, \ldots, b_d)$. This is the function used on Figure 14.

$$\text{AMD}_k(b_1, \ldots, b_d) = f_k(q_{b_1,\ldots,b_\sigma}, q_{b_{\sigma+1},\ldots,b_{2\sigma}}, \ldots, q_{b_{d-\sigma+1},\ldots,b_d}).$$

The definition above assumes without loss of generality that d is a multiple of σ (otherwise, the can pad $b_1 \ldots b_d$ with enough zeroes).

3

Outsourcing MPC Precomputation for Location Privacy

Ivan Oleynikov, Elena Pagnin, Andrei Sabelfeld

EuroS&P Location-Privacy Workshop'2022

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.

This page intentionally left blank.

3.1 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 [9], 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 [9, 18, 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):

$$f((x_A, y_A), (x_B, y_B)) = \begin{cases} 1, & \text{if } (x_A - x_B)^2 + (y_A - y_B)^2 \leq R^2, \\ 0, & \text{otherwise.} \end{cases} \tag{3.1}$$

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 (3.1). 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 [9, 18, 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 [3], SPDZ [9] and edaBits [2] 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 [7] 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 argue its security in Canetti's hybrid model [4]. In addition, 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 $\text{ABY}^{\text{C}}_{\text{AY}}$ [9] 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 $\text{ABY}^{\text{C}}_{\text{Y}}$.

3.2 Preliminaries

The clients in POLAR run a combination of MPC protocols in order to implement the PT functionality (Equation 3.1). 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 3.1) 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 [2] 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 18. This functionality can be implemented by the edaBits [2] 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 [2]). 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 [2].

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., $[\![x]\!]_p \leftarrow \text{ConvertB2A}([\![y]\!]_2)$ means sending $(\text{ConvertB2A}, \text{“}x\text{”}, \text{“}y\text{”})$ 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

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

In real-life, the $\mathcal{F}_{\text{AB-MPC}}$ will be implemented by a combination of edaBits, SDPZ [9] and Tinier [3]. 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 19. The precomputed data for the clients generated by this functionality consists of SPDZ [9] multiplication triples, Tinier [3] multiplication triples, random Tinier shares (to allow the clients to input values into Tinier), daBits [34] and edaBits [2]. In practice, the $\mathcal{F}_{\text{Out-MPC}}$ functionality will be implemented by applying the outsourcing technique [7] 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, because it is trivial and unnecessarily technical for our presentation. Full details can be found in the source code of our implementation.

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

Input: On input $(\text{Input}, P_i, \text{type}, \text{id}, x)$ from P_i and $(\text{Input}, P_i, \text{type}, \text{id})$ from all other parties, with id a fresh identifier, type $\in \{\text{binary}, \text{arithmetic}\}$ 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 $(\text{LinComb}, \text{type}, \text{id}, (\text{id}_i)_{i=1}^m, (c_j)_{j=0}^m)$, 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 $((\text{type}, \text{id}_1, x_1), \ldots (\text{type}, \text{id}_m, x_m))$, 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 $(\text{Mult}, \text{type}, \text{id}, \text{id}_1, \text{id}_2)$ 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 $(\text{ConvertB2A}, \text{id}, \text{id}')$ from all parties, retrieve $(\text{binary}, \text{id}', x)$ and store $(\text{arithmetic}, \text{id}, x)$.

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

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

Output: On input $(\text{Output}, \text{type}, \text{id})$ 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 18: Ideal functionality $\mathcal{F}_{\text{AB-MPC}}$ of MPC arithmetic blackbox modulo 2 and modulo p [2]

3.3 The POLAR Protocol

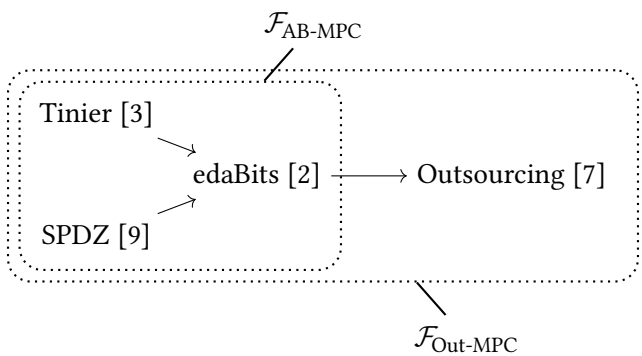


Figure 20: The diagram of blackbox applications of previous works that yields the functionalities that we use in our hybrid model.

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_{Alice} and Bob's data as z_{Bob}. Output Eval to the adversary. If Alice and Bob are corrupted, deliver the corresponding $z_{\cdot}$ to the corrupted clients.

Output. On command (Deliver, C) from the adversary where C is either Alice or Bob, deliver z_C to the corresponding client.

Figure 19: 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 [7, Figure 3].

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 19) and $\mathcal{F}_{\text{AB-MPC}}$ (Figure 18). In practice, the two ideal functionalities will be replaced by the corresponding protocols that implement them [2, 7] 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 3.1). 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 20 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.

POLAR involves four parties (Figure 21): 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 22 shows the formal definition of the ideal functionality $\mathcal{F}_{\text{PT}}$ that POLAR implements, while Figure 23 shows how POLAR implements $\mathcal{F}_{\text{PT}}$ in our hybrid setting.

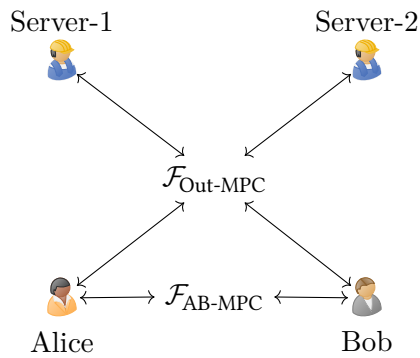


Figure 21: Diagram of the hybrid setting we describe our protocol POLAR in.

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

Figure 19 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 3.1), 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
$$[\![\rho]\!]_2 \leftarrow \text{Compare}([\![D]\!]_p, R^2).$$
6. The clients output the $[\![\rho]\!]_2$ to Alice.

Figure 23: The POLAR protocol

3.4 Security Analysis

To prove the security of POLAR (Figure 23) we show that it securely implements the functionality $\mathcal{F}_{\text{PT}}$ (Figure 22) 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 2.

Theorem 2. *The protocol POLAR* securely computes $\mathcal{F}_{\text{PT}}$ 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 20), 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 [2, 3, 7, 9] and check the details). The main objective of this section is to show that the techniques from Figure 20 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 18) 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 2, which requires one of the servers to stay uncorrupted.

3.5 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 $\text{ABY}^{\text{C}}_{\text{Y}}$ [9], 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.

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

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 [9]: `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.

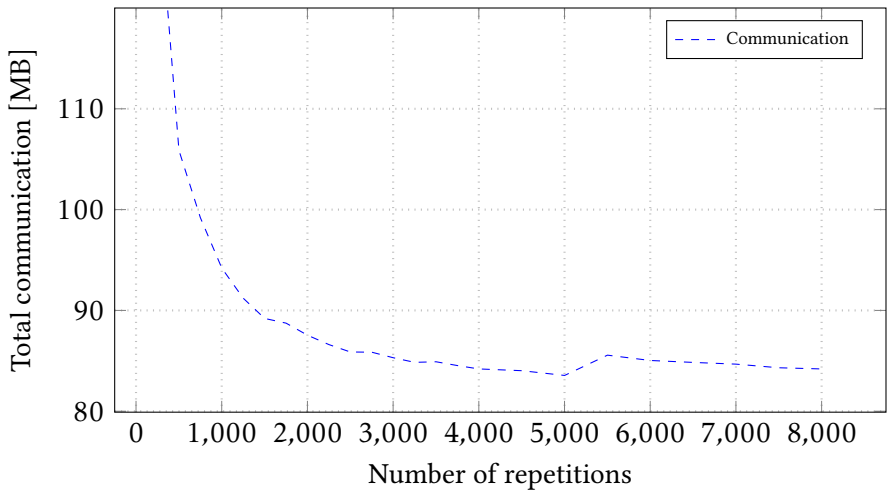
For the $\mathcal{F}_{\text{AB-MPC}}$ that is executed by the servers (as part of $\mathcal{F}_{\text{Out-MPC}}$), the SPDZ plaintext values are 64 bits (modulo a prime), and all the other parameters are as above. The statistical security parameter for edaBits is 40, the bucket size is $B = 4$. The preprocessing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

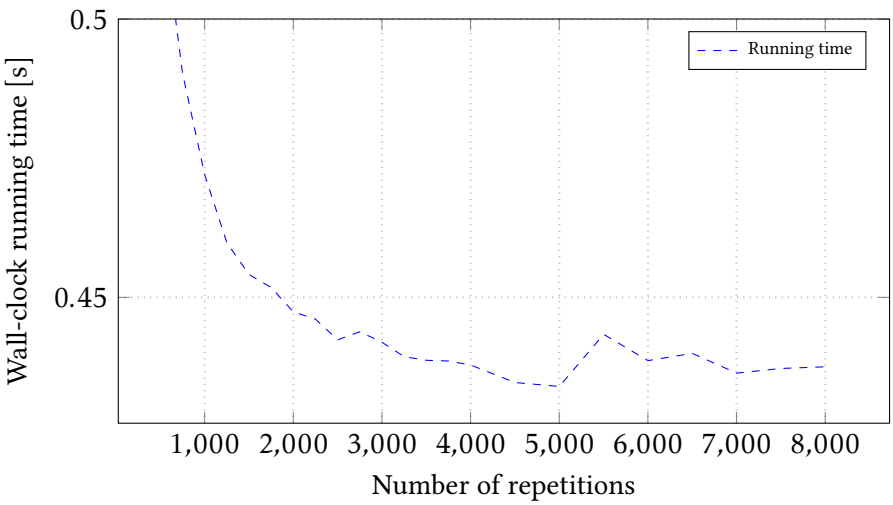
Figure 24 shows the amortized performance of servers in POLAR depending on the number of times the protocol is repeated. These measurements include both setup time and the actual protocol execution. As the number of repetitions approaches 5000, the amortized execution time reaches 0.4 seconds, and the 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

speculate was caused by some internal details of MP-SPDZ library which we use. 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 ra-



(a) Total communication



(b) Running time

Figure 24: Amortized performance of servers in POLAR by the number of repetitions

dius R, this is reflected in the measurements presented on Figure 25. 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 secure, and in OLIC the servers work with client data directly).

3.6 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

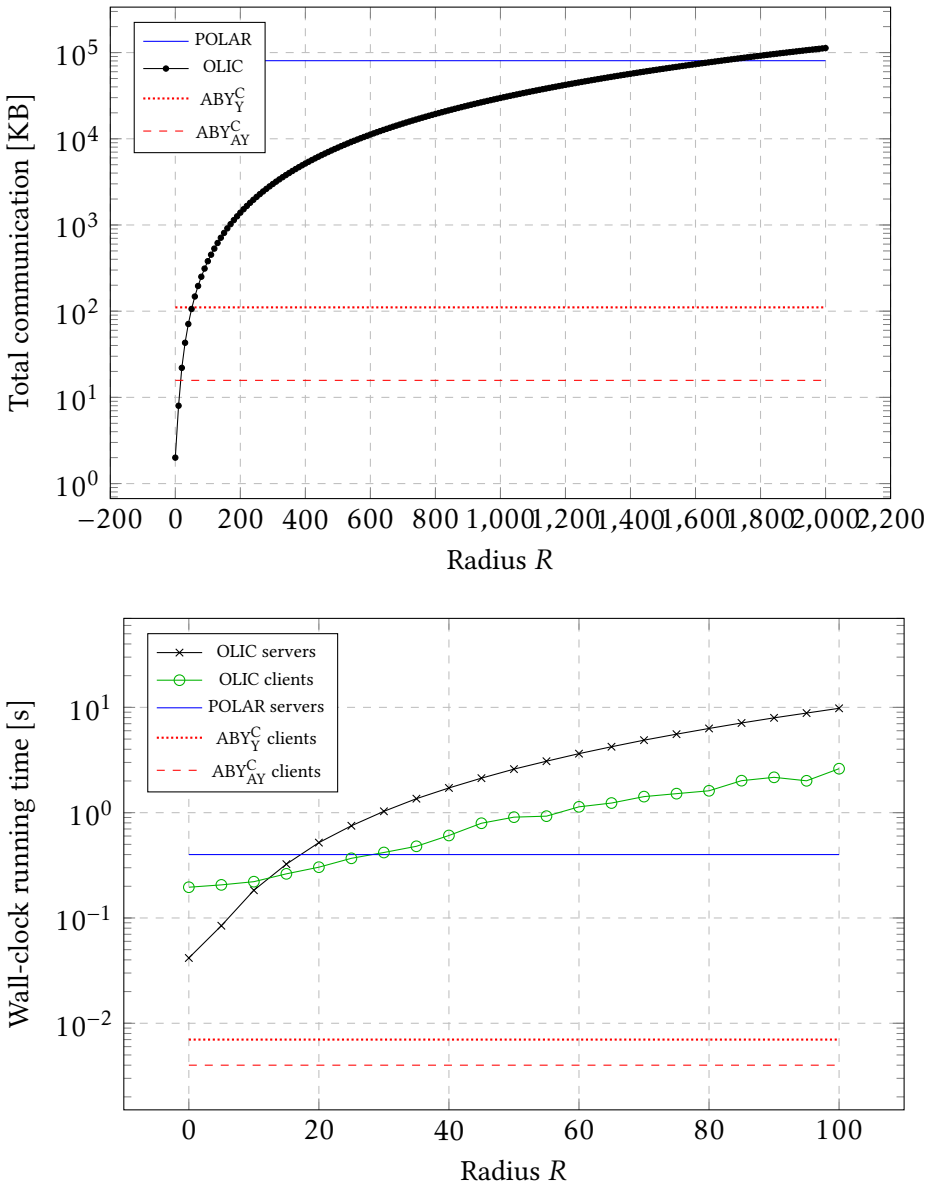


Figure 25: Comparison of POLAR with OLIC, ABY^C_Y and ABY^C_{AY}

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. [9] 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 [3] MPC protocols for computations in the two different domains, and the edaBits [2] 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 [9, 13, 18, 19, 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.

3.7 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 [9, 18, 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 a 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.

3.8 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 eval-

uation 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).

This page intentionally left blank.

Bibliography

[1] A. Beimel, A. Gabizon, Y. Ishai, E. Kushilevitz, S. Meldgaard, and A. Paskin-Cherniavsky. Non-interactive secure multiparty computation. In J. A. Garay and R. Gennaro, editors, *CRYPTO*, volume 8617 of *LNCS*, pages 387–404. Springer, 2014.

[2] F. Benhamouda, H. Krawczyk, and T. Rabin. Robust non-interactive multiparty computation against constant-size collusion. In J. Katz and H. Shacham, editors, *CRYPTO*, volume 10401 of *LNCS*, pages 391–419. Springer, 2017.

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

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

[5] C. Castelluccia, N. Bielova, A. Boutet, M. Cunche, C. Lauradoux, D. Le Métayer, and V. Roca. Robert: Robust and privacy-preserving proximity tracing. 2020.

[6] S. 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] J. 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] J. Cox and M. 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] I. Damgard, V. Pastro, N. Smart, and S. Zakarias. Multiparty computation from somewhat homomorphic encryption. In R. Safavi-Naini and R. Canetti, editors, *Advances in Cryptology – CRYPTO 2012*, pages 643–662, Berlin, Heidelberg, 2012. Springer Berlin Heidelberg.

[10] D. Demmler, T. Schneider, and M. 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] D. Escudero, S. Ghosh, M. Keller, R. Rachuri, and P. Scholl. Improved primitives for mpc over mixed arithmetic-binary circuits. Cryptology ePrint Archive, Report 2020/338, 2020. https://eprint.iacr.org/2020/338.

[12] T. K. Frederiksen, M. Keller, E. Orsini, and P. Scholl. A unified approach to mpc with preprocessing using ot. Cryptology ePrint Archive, Report 2015/901, 2015. https://eprint.iacr.org/2015/901.

[13] D. Freni, C. R. Vicente, S. Mascetti, C. Bettini, and C. S. Jensen. Preserving location and absence privacy in geo-social networks. In *CIKM*, pages 309–318, 2010.

[14] S. D. Gordon, T. Malkin, M. Rosulek, and H. Wee. Multi-party computation of polynomials and branching programs without simultaneous interaction. In T. Johansson and P. Q. Nguyen, editors, *EUROCRYPT*, LNCS. Springer, 2013.

[15] S. Halevi, Y. Ishai, A. Jain, I. Komargodski, A. Sahai, and E. Yogev. Non-interactive multiparty computation without correlated randomness. In T. Takagi and T. Peyrin, editors, *ASIACRYPT*, volume 10626 of *LNCS*, pages 181–211. Springer, 2017.

[16] S. Halevi, Y. Lindell, and B. 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] P. A. Hallgren, M. Ochoa, and A. Sabelfeld. InnerCircle: A parallelizable decentralized privacy-preserving location proximity protocol. In *PST*, pages 1–6, 2015.

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

[20] A. 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] T. P. Jakobsen, J. B. Nielsen, and C. 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] A. Jarrous and B. Pinkas. Canon-mpc, a system for casual non-interactive secure multi-party computation using native client. In A. Sadeghi and S. Foresti, editors, *WPES*, pages 155–166. ACM, 2013.

[23] K. Järvinen, A. Kiss, T. Schneider, O. Tkachenko, and Z. Yang. Faster privacy-preserving location proximity schemes for circles and polygons. *IET Information Security*, 14, 10 2019.

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

[25] M. Keller, E. Orsini, and P. 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] P. Kotzanikolaou, C. Patsakis, E. Magkos, and M. Korakakis. Lightweight private proximity testing for geospatial social networks. *Computer Communications*, 73:263–270, 2016.

[27] Y. 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] 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.

[29] A. Narayanan, N. Thiagarajan, M. Lakhani, M. Hamburg, and D. Boneh. Location privacy via private proximity testing. In *NDSS*, 2011.

[30] J. D. Nielsen, J. I. Pagter, and M. B. Stausholm. Location privacy via actively secure private proximity testing. In *PerCom Workshops*, pages 381–386. IEEE CS, 2012.

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

[32] E. Pagnin, G. Gunnarsson, P. Talebi, C. Orlandi, and A. Sabelfeld. TOP-Pool: Time-aware Optimized Privacy-Preserving Ridesharing. *PoPETs*, 2019(4):93–111, 2019.

[33] 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*, 2015.

[34] D. Rotaru and T. 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] M. N. Sakib and C. Huang. Privacy preserving proximity testing using elliptic curves. In *ITNAC*, pages 121–126. IEEE Computer Society, 2016.

[36] G. Saldamli, R. Chow, H. Jin, and B. P. Knijnenburg. Private proximity testing with an untrusted server. In *WISEC*, pages 113–118. ACM, 2013.

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

[38] L. Siksnys, J. R. Thomsen, S. Saltenis, and M. L. Yiu. Private and flexible proximity detection in mobile social networks. In *MDM*, pages 75–84, 2010.

[39] L. Siksnys, J. R. Thomsen, S. Saltenis, M. L. Yiu, and O. Andersen. A location privacy aware friend locator. In *SSTD*, pages 405–410, 2009.

[40] C. Troncoso, M. Payer, J.-P. Hubaux, M. Salathé, J. Larus, E. Bugnion, W. Lueks, T. Stadler, A. Pyrgelis, D. Antonioli, et al. Decentralized privacy-preserving proximity tracing. *arXiv preprint arXiv:2005.12273*, 2020.

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