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

Ivan Oleynikov[1], Elena Pagnin[2] and Andrei Sabelfeld[1]
[1] *Chalmers University, Gothenburg, Sweden*
[2] *Lund University, Lund, Sweden*
{andrei,ivanol}@chalmers.se, elena.pagnin@eit.lth.se

Keywords:
Privacy, Proximity-Testing, Multi-Party Computation, Active Security.

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.

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 [Troncoso et al., 2020] (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 [Cox, 2019], and similar cases were reported about Uber [Hern, 2016], Yahoo [Cole, 2019], and Facebook [Cox and Hoppen-

stedt, 2018]. 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 (provide 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 [Zhong et al., 2007,Siksnys et al., 2009,Siksnys et al., 2010,Freni et al., 2010,Mascetti et al., 2011,Narayanan et al., 2011, Nielsen et al., 2012, Kotzanikolaou et al., 2016] 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 [Järvinen et al., 2019], 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 [Hallgren et al., 2015,Oleynikov et al., 2020,Järvinen et al., 2019, Pagnin et al., 2019], 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 [Hallgren et al., 2015, Järvinen et al., 2019, Oleynikov et al., 2020] on Euclidean distance-based PT:

`Offline` We adopt the setting of "napping party" [Oleynikov et al., 2020]: 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 [Hallgren et al., 2015,Oleynikov et al., 2020] 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. [Oleynikov et al., 2020], 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 [Zhong et al., 2007,Hallgren et al., 2015,Oleynikov et al.,

2020, Hallgren et al., 2016, Sakib and Huang, 2016, Järvinen et al., 2019] 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 1 summarizes the features achieved by our protocol, CatNap, compared to the most relevant recent works. The InnerCircle protocol by Hallgren et al. [Hallgren et al., 2015] 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. [Järvinen et al., 2019] 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. [Oleynikov et al., 2020] 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 [Hallgren et al., 2015]: passive security and R^2-dependent performance. These works are further discussed in section 5. This paper presents CatNap, the first protocol for privacy-enhancing location PT to achieve all the above three properties.

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 [Canetti, 1998], 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 [Hallgren et al., 2015], OLIC [Oleynikov et al., 2020], $\text{ABY}^{\text{C}}_{\text{Y}}$ and $\text{ABY}^{\text{C}}_{\text{AY}}$ [Järvinen et al., 2019]. Although the InnerCircle, $\text{ABY}^{\text{C}}_{\text{AY}}$, and $\text{ABY}^{\text{C}}_{\text{Y}}$ protocols [Hallgren et al., 2015, Järvinen et al., 2019] 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.

Table 1: Comparison of CatNap features to the related protocols

Protocol	`Offline`	`RadiusInd`	`ActiveSec`
InnerCircle [Hallgren et al., 2015]	−	−	−
$\text{ABY}^{\text{C}}_{\text{Y}}$ and $\text{ABY}^{\text{C}}_{\text{AY}}$ [Järvinen et al., 2019]	−	+	−
OLIC [Oleynikov et al., 2020]	+	−	−
CatNap	+	+	+

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 [Escudero et al., 2020], SPDZ2k [Cramer et al., 2018], Tinier [Frederiksen et al., 2015] 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 [Keller, 2020]. In our protocol, the clients "outsource" the functionality computation to the servers using the technique of Jakobsen et. al. [Jakobsen et al., 2014]: 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 [Cramer et al., 2018] and Tinier [Frederiksen et al., 2015], 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 [Rotaru and Wood, 2019] and edaBits [Escudero et al., 2020] 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 [Lindell, 2016], the same setting was used for a number of previous PT protocols [Hallgren et al., 2015, Oleynikov et al., 2020, Järvinen et al., 2019] 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 subroutine 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 [Hallgren et al., 2016] 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 [Sharemind, 2022] framework.

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 1. This functionality is implemented by the edaBits [Escudero et al., 2020] 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 [Escudero et al., 2020]). For the sake of generality, we define `Compare` as a standalone command and leave its specification up to specific implementations. The edaBits [Escudero et al., 2020] is implemented in MP-SPDZ [Keller, 2020] 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 = `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., $[\![x]\!]_{2^m} \leftarrow$ `ConvertB2A`$([\![y]\!]_2)$ means sending (`ConvertB2A`, "x", "y") 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 $\mathcal{F}_2$), 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

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

Linear Combination: On input $(\texttt{LinComb}, \text{type}, \text{id}, (\text{id}_i)_{i=1}^m, (c_j)_{j=0}^m)$, where each id_j is stored in memory and $c_j \in \mathbb{Z}_2$ if type $= \texttt{binary}$ or $c_j \in \mathbb{Z}_{2^k}$ if type $= \texttt{arithmetic}$, 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 $= \texttt{binary}$ and modulo 2^k if type $= \texttt{arithmetic}$, and store $(\text{type}, \text{id}, y)$.

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

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

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

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

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

Figure 1: Ideal functionality $\mathcal{F}_{\text{AB-MPC}}$ of MPC arithmetic blackbox modulo 2 and modulo 2^k [Escudero et al., 2020]

a vector of bits $([\![v_0]\!]_2, \ldots [\![v_{k-1}]\!]_2)$, 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 $[\![\overrightarrow{u}]\!]_2 = ([\![u_0]\!]_2, \ldots [\![u_{k-1}]\!]_2)$ and $[\![\overrightarrow{v}]\!]_2 = ([\![v_0]\!]_2, \ldots [\![v_{k-1}]\!]_2)$, 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

$$
\begin{aligned}
[\![p_0]\!]_2 &\leftarrow \texttt{Mult}(\texttt{binary}, [\![u_0]\!]_2, [\![v_0]\!]_2) \\
&\ldots \\
[\![p_{k-1}]\!]_2 &\leftarrow \texttt{Mult}(\texttt{binary}, [\![u_{k-1}]\!]_2, [\![v_{k-1}]\!]_2) \\
c &\leftarrow (0, 1, \ldots, 1) \in \mathcal{F}_2^{k+1}
\end{aligned}
$$

(The next line is computing the sum of all $[\![p_i]\!]_2$)

$$
[\![b]\!]_2 \leftarrow \texttt{LinComb}(\texttt{binary}, [\![\overrightarrow{p}]\!]_2, c).
$$

3 The CatNap Protocol

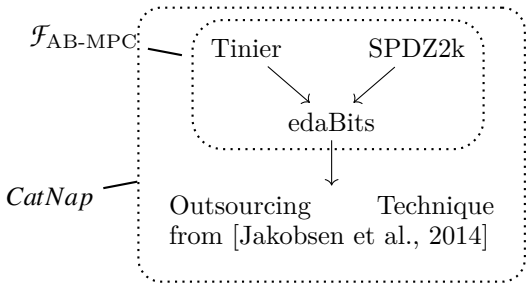


Figure 2: The diagram of blackbox applications of previous works that yields CatNap protocol and the $\mathcal{F}_{\text{AB-MPC}}$ functionality that we use to build CatNap.

The CatNap protocol is built by combining the previous works in a blackbox way, i.e., relying only on their most standard properties. Figure 2 gives an 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 2 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.

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 ex-

ecution, 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 3 shows the formal definition of the ideal functionality $\mathcal{F}_{\text{PT}}$ that CatNap implements, while Figure 5 shows how CatNap implements $\mathcal{F}_{\text{PT}}$ in the real world.

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

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 [Jakobsen et al., 2014]. 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:

$$D = (x_a - x_b)^2 - (y_a - y_b)^2. \quad (1)$$

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 [Jakobsen et al., 2014]. 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 the $\mathcal{F}_{\text{AB-MPC}}$ functionality.

The transferring of client inputs into the $\mathcal{F}_{\text{AB-MPC}}$ functionality mentioned above is done in their bit-decomposed form: each coordinate is represented as a bit-vector of fixed length, the vectors of all coordinates are concatenated together and the resulting vector is transferred (using a special subprotocol `Transfer`, shown below) into the $\mathcal{F}_{\text{AB-MPC}}$. This has a useful side-effect: we can naturally bound inputs provided by each client by limiting the number of bits used to represent them (since the `Transfer` accepts only fixed number of bits). This way, a malicious client cannot input values that are too large and may cause an overflow modulo 2^m in the computation of D (see Equation (1)). We limit each client coordinate to k bits, where k can be any positive integer value such that $2k + 3 \leq m$ (this ensures that there is no overflow in the expression for D from Equation (1)). In other words, for all meaningful values of R, it must hold that $0 \leq R \leq 2^k\sqrt{2}$.

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 1).
Initial condition: The client knows its input $z \in \mathcal{F}_2^l$, 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) $\kappa \leftarrow^{\$} \mathcal{F}_{2^\sigma}$
   (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) $r^{(1)} \leftarrow^{\$} \mathcal{F}_{2^l}$
   (b) $r^{(2)} = z \oplus r^{(1)}$
   (c) $\kappa^{(2)} = \kappa \oplus \kappa^{(1)}$
   (d) $t^{(1)} \leftarrow^{\$} \mathcal{F}_{2^\sigma}$
   (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) $[\![r_i^{(1)}]\!]_2 \leftarrow \texttt{Input}_{\texttt{Server-1}}(r_i^{(1)})$ for $i \in \{0 \dots l-1\}$
   (b) $[\![r_i^{(2)}]\!]_2 \leftarrow \texttt{Input}_{\texttt{Server-2}}(r_i^{(2)})$ for $i \in \{0 \dots l-1\}$
   (c) $[\![t_i^{(1)}]\!]_2 \leftarrow \texttt{Input}_{\texttt{Server-1}}(t_i^{(1)})$ for $i \in \{0 \dots \sigma-1\}$
   (d) $[\![t_i^{(2)}]\!]_2 \leftarrow \texttt{Input}_{\texttt{Server-2}}(t_i^{(2)})$ 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 $c \leftarrow \texttt{Output}([\![c]\!]_2)$ 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 4: The `Transfer` sub-protocol

The `Transfer` Sub-protocol. Figure 4 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 $z \in \mathcal{F}_2^l$ 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 5).

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 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 5 provides a detailed overview of our CatNap protocol. We recall that CatNap implements the $\mathcal{F}_{\text{PT}}$ functionality from Figure 3. 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.

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 1). 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 4) 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
   $$[\![x_a]\!]_{2^m} \leftarrow \texttt{ConvertB2A}([\![\overrightarrow{x_a}]\!]_2)$$
   $$[\![y_a]\!]_{2^m} \leftarrow \texttt{ConvertB2A}([\![\overrightarrow{y_a}]\!]_2)$$
   $$[\![x_b]\!]_{2^m} \leftarrow \texttt{ConvertB2A}([\![\overrightarrow{x_b}]\!]_2)$$
   $$[\![y_b]\!]_{2^m} \leftarrow \texttt{ConvertB2A}([\![\overrightarrow{y_b}]\!]_2).$$
   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) $[\![\rho]\!]_2 \leftarrow \texttt{Compare}([\![D]\!]_{2^m}, R^2)$.
3. The servers mask the bit ρ with μ and reveal the result:
   (a) $[\![\rho']\!]_2 \leftarrow [\![\rho]\!]_2 \oplus [\![\mu]\!]_2$
   (b) $\rho' \leftarrow \texttt{Output}([\![\rho']\!]_2)$.
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 5: The CatNap protocol

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).

3.1 Security Proof

To prove the security of CatNap (Figure 5) we must show that it securely implements [Lindell, 2016] the functionality $\mathcal{F}_{\text{PT}}$ (Figure 1) in the presence of static active adversary who can corrupt any subset of parties as long as one of the servers is not corrupted.

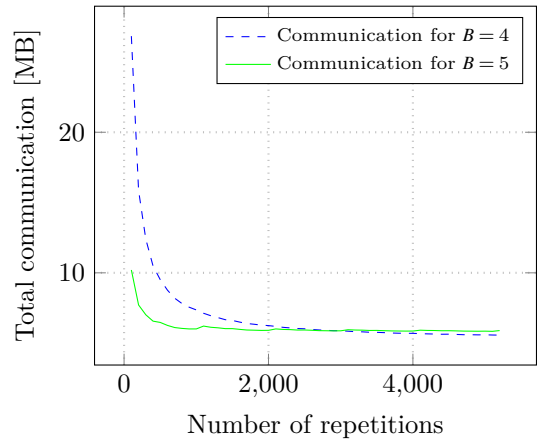
Since the protocol CatNap is modular and is built from off-the-shelf MPC techniques (shown on Figure 2), we prove its security by combining the proofs of the corresponding techniques. Showing here all the details would be too technical and not very insightful, therefore we only give an overview of the major steps (but we encourage a curious reader to go through the formal definitions of the used techniques [Frederiksen et al., 2015, Cramer et al., 2018, Escudero et al., 2020, Jakobsen et al., 2014] and check the details). The main objective of this section is to show that the techniques from Figure 2 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 [Keller, 2020] framework.

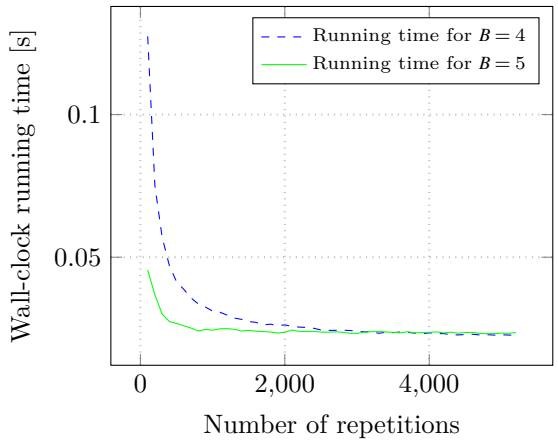
Combination of edaBits with the outsourcing of computation technique is not as straightforward: outsourcing of computation can be applied to either an arithmetic MPC or a binary one, but edaBits (which implements the ideal functionality $\mathcal{F}_{\text{AB-MPC}}$ shown on Figure 1) combines both. 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 4 and 5) on $\mathcal{F}_{\text{AB-MPC}}$ work with bits while all the arithmetic operations are done only inside $\mathcal{F}_{\text{AB-MPC}}$.

4 Evaluation

To evaluate the performance of CatNap, we implemented it in the MP-SPDZ [Keller, 2020] cryptographic framework and made it available online [Oleynikov et al., 2022]. 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.



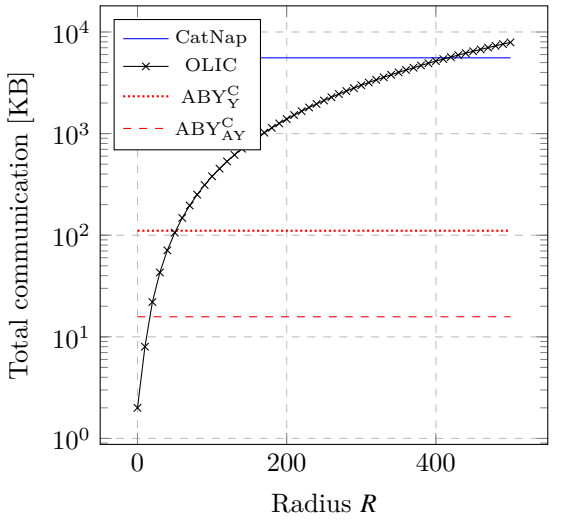
(a) Total communication



(b) Running time

Figure 6: Amortized performance of CatNap by the number of repetitions

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 [Demmler et al., 2015] 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.



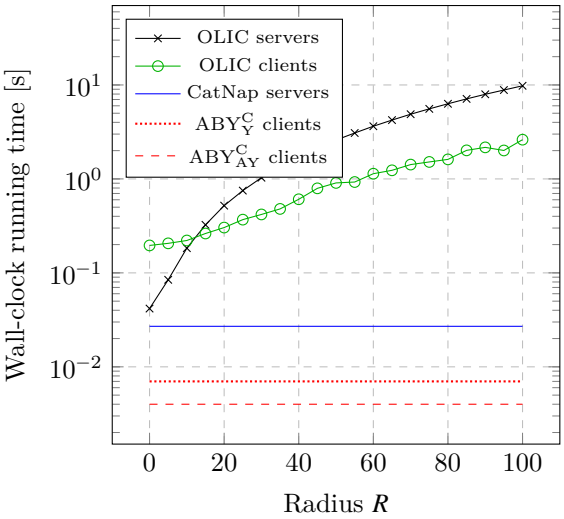


Figure 7: Comparison of CatNap with OLIC, $\text{ABY}^{\text{C}}_{\text{Y}}$ and $\text{ABY}^{\text{C}}_{\text{AY}}$

OLIC. We use the most efficient one of the two instantiations presented in the original paper [Oleynikov et al., 2020], namely, the **(EC)** which is based on Curve25519 and M383 elliptic curves.

$\mathbf{ABY^C_{AY}}$ and $\mathbf{ABY^C_Y}$. We use ABY [Demmler et al., 2015] parameters of the original paper [Järvinen et al., 2019]: `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 5 and 4, 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 6 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 7. The protocols that have performance independent of R are shown there as straight horizontal lines. Notably, CatNap is less efficient than $\mathrm{ABY^C_{AY}}$ and $\mathrm{ABY^C_Y}$ (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.

5 Related Work

Zhong et al. [Zhong et al., 2007] 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. [Narayanan et al., 2011] 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. [Saldamli et al., 2013] build on the protocol with the oblivious server and suggest optimizations based on properties from geometry and linear algebra. Nielsen et al. [Nielsen et al., 2012] and Kotzanikolaou et al. [Kotzanikolaou et al., 2016] also propose grid-based solutions.

Hide&Crypt by Freni et al. [Freni et al., 2010] 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. [Mascetti et al., 2011] 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. [Siksnys et al., 2009] 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. [Siksnys et al., 2010] 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 [Sedenka and Gasti, 2014] 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. [Hallgren et al., 2015] introduce InnerCircle for parallelizable decentralized PT, using additively homomorphic encryption between two parties that must be online. The MaxPace [Hallgren et al., 2016] protocol builds on the speed constraints of an InnerCircle-style

protocol as to limit the effects of trilateration attacks. Polakis [Polakis et al., 2015] study different distance and proximity disclosure strategies employed in the wild and experiment with practical effects of trilateration.

Sakib and Huang [Sakib and Huang, 2016] explore PT using elliptic curves. They require Alice and Bob to be online to be able to run the protocol. Järvinen et al. [Järvinen et al., 2019] 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. [Hallgren et al., 2017] 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. [Oleynikov et al., 2020] 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. [Halevi et al., 2011], using one server for this purpose will leak the clients' data to it. Further works on generic MPC in client-server settings [Jarrous and Pinkas, 2013, Gordon et al., 2013, Halevi et al., 2017, Beimel et al., 2014, Benhamouda et al., 2017] also consider one-server scenarios. Some of these protocols are mentioned in Table 1.

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 [Cramer et al., 2018], Tinier [Frederiksen et al., 2015] for Boolean computation and edaBits [Escudero et al., 2020] 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 [Troncoso et al., 2020]. 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 [Zhong et al., 2007, Siksnys et al., 2009, Siksnys et al., 2010, Freni et al., 2010, Narayanan et al., 2011, Saldamli et al., 2013, Sedenka and Gasti, 2014, Hallgren et al., 2015, Oleynikov et al., 2020, Hallgren et al., 2016, Sakib and Huang, 2016, Järvinen et al., 2019] 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. [Mascetti et al., 2011], 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 [Zhong et al., 2007, Siksnys et al., 2009, Siksnys et al., 2010, Freni et al., 2010, Mascetti et al., 2011, Narayanan et al., 2011, Nielsen et al., 2012, Kotzanikolaou et al., 2016] resort to grid-based approximations, thus losing precision of proximity tests.

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 [Pagnin et al., 2019], 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 [Bla, 2022]), 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 blackbox 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).

REFERENCES

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

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

Bla (2022). BlaBlaCar - Trusted carpooling. `https://www.blablacar.com/`.

Canetti, R. (1998). Security and composition of multi-party cryptographic protocols. ePrint. `https://eprint.iacr.org/1998/018`.

Cole, S. (2019). 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`. [Online; accessed 22-Mar-2022].

Cox, J. (2019). 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`. [Online; accessed 22-Mar-2022].

Cox, J. and Hoppenstedt, M. (2018). Sources: Facebook has fired multiple employees for snooping on users. `https://www.vice.com/en/article/bjp9zv/facebook-employees-look-at-user-data`. [Online; accessed 22-Mar-2022].

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

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

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

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

Freni, D., Vicente, C. R., Mascetti, S., Bettini, C., and Jensen, C. S. (2010). Preserving location and absence privacy in geo-social networks. In *CIKM*.

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

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

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

Hallgren, P., Orlandi, C., and Sabelfeld, A. (2017). PrivatePool: Privacy-Preserving Ridesharing. In *CSF*.

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

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

Hern, A. (2016). Uber employees 'spied on ex-partners, politicians and beyoncé'. `https://www.theguardian.com/technology/2016/dec/13/uber-employees-spying-ex-partners-politicians-beyonce`. [Online; accessed 22-Mar-2022].

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

Jarrous, A. and Pinkas, B. (2013). Canon-mpc, a system for casual non-interactive secure multiparty computation using native client. In *WPES*. ACM.

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

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

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

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

Mascetti, S., Freni, D., Bettini, C., Wang, X. S., and Jajodia, S. (2011). Privacy in geo-social networks: proximity notification with untrusted service providers and curious buddies. *VLDB J.*

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

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

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

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

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

Polakis, I., Argyros, G., Petsios, T., Sivakorn, S., and Keromytis, A. D. (2015). Where's wally?: Precise user discovery attacks in location proximity services. In *CCS*.

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

Sakib, M. N. and Huang, C. (2016). Privacy preserving proximity testing using elliptic curves. In *ITNAC*.

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

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

Sharemind (2022). Sharemind MPC Platform. `https://sharemind.cyber.ee/sharemind-mpc/multi-party-computation/`.

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

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

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

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

A Algebraic Manipulation Detection Code

In this section, we define the Algebraic Manipulation Detection (AMD) code employed in the outsourcing technique [Jakobsen et al., 2014] 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 ([Jakobsen et al., 2014])**.** *The function $f_k(x_1,\dots x_l) = t$, where $k, x_i, t \in \mathcal{F}$ for some finite field $\mathcal{F}$, 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. [Jakobsen et al., 2014] 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 $\log|\mathcal{F}|$ 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 $\mathcal{F}$ to authenticate them. Here, we also use a trick from Jakobsen et. al.: take $\mathcal{F}$ to be $\mathcal{F}_{2^\sigma} = \mathcal{F}_2[x]/(p(x))$, i.e., the field of polynomials with coefficients from $\mathcal{F}_2 = \{0,1\}$ where all the field operations are performed modulo an irreducible polynomial $p(x)$ with $\deg p = \sigma$. Any vector $v = (v_0,\dots,v_{\sigma-1})$ 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 1) 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 $\text{AMD}_k(b_1,\dots,b_d)$. This is the function used on Figure 4.

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

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