Inference Attacks and Defenses in Continuous Location Sharing with Local Differential Privacy

by

Muhammed Esad Simitcioğlu

A Dissertation Submitted to the
Graduate School of Sciences and Engineering
in Partial Fulfillment of the Requirements for
the Degree of

Master of Science

in

Computer Science and Engineering

KOÇ ÜNİVERSİTESİ

August 1, 2025

Inference Attacks and Defenses in Continuous Location Sharing with Local Differential Privacy

Koç University

Graduate School of Sciences and Engineering

This is to certify that I have examined this copy of a master's thesis by

Muhammed Esad Simitcioğlu

and have found that it is complete and satisfactory in all respects,
and that any and all revisions required by the final
examining committee have been made.

Committee Members:

_____

Assist. Prof. Mehmet Emre Gürsoy (Advisor)

_____

Prof. Alptekin Küpçü

_____

Prof. Yücel Saygın

Date: _____

ABSTRACT

Inference Attacks and Defenses in Continuous Location Sharing with Local Differential Privacy
Muhammed Esad Simitcioğlu
Master of Science in Computer Science and Engineering
August 1, 2025

Local differential privacy (LDP) has recently emerged as a commonly used privacy standard. With the growing popularity of LDP, recent works started applying LDP to location data collection and privacy-preserving usage of location-based services (LBS). However, in many practical applications of location data sharing and LBS usage, the user must continuously share his/her location with the data collector (e.g., LBS provider). Although the privacy of each individual location is protected with LDP, correlations between the user's consecutive locations can be exploited to infer the user's true locations.

Following this idea, in this thesis we propose novel inference attacks against continuous location sharing under LDP. We develop attacks in two categories: statistical attacks based on Bayesian adversary formulation for stationary users and Hidden Markov Model (HMM) based attacks for mobile users. We also propose two extensions for our HMM-based attacks (informed and chain attacks), which enable the attacker to incorporate aggregate population statistics or use multiple iterations of HMM construction to improve attack effectiveness. We adapt and apply our attacks to four popular LDP protocols (GRR, RAPPOR, OUE, OLH), three datasets, and varying privacy levels. Experiment results show that our attacks are highly effective, which highlights the privacy risks of using LDP in continuous location sharing. Furthermore, results show that the attacks are better suited for the types of users they were designed for, i.e., statistical attacks achieve higher effectiveness than HMM-based attacks in case of stationary users, whereas HMM-based attacks achieve higher effectiveness in case of mobile users.

Although our HMM-based attacks are effective on bitvector-based LDP protocols (such as RAPPOR and OUE), we observe that they suffer from efficiency problems, i.e., high memory usage and execution time. These hinder the scalability of our

attacks, especially when grid sizes are large or when the attacker aims to achieve precise location inference. We therefore propose heuristic methods to improve the time and memory-efficiency of our attacks by removing states with low occurrence probability from the attack HMMs. We experimentally show that our improvements can reduce the memory and time costs of the attacks by more than an order of magnitude, while maintaining similar effectiveness compared to their original versions.

Finally, we propose three defense strategies against our attacks: Memoization, Replay, and Replication. In memoization, each user stores a perturbed version of their original value, which is perturbed for a second time before being reported to the data collector. In replay, each user maintains a cache of previous locations and their perturbed counterparts. When a certain location is repeated, the user replays the old perturbed version rather than performing a fresh perturbation. In replication, each user replicates their previously reported value if their current location is similar to their previous location. We evaluate the three defenses using the same experiment setup containing four LDP protocols, three datasets, varying privacy levels, and mobile and stationary users. Results show that our defenses are successful in reducing attack effectiveness. We critically analyze the success, efficiency, and utility aspects of the three defenses under varying conditions, and provide recommendations regarding when to use which defense.

ÖZETÇE

Lokal Diferansiyel Mahremiyetli Sürekli Konum Paylaşımında Çıkarım Saldırıları ve Savunmalar
Muhammed Esad Simitcioğlu
Bilgisayar Bilimi ve Mühendisliği, Yüksek Lisans
1 Ağustos 2025

Lokal diferansiyel mahremiyet (LDP), son zamanlarda yaygın olarak kullanılan bir mahremiyet standardı haline gelmiştir. LDP'nin artan popülerliği ile birlikte, güncel çalışmalar LDP'yi konum verilerinin toplanması ve konum tabanlı hizmetlerin (LBS) mahremiyet korumalı kullanımı üzerinde uygulamaya başlamıştır. Ancak, konum verisi paylaşımı ve LBS kullanımının birçok pratik uygulamasında, kullanıcının konumunu veri toplayıcı (örneğin LBS sağlayıcısı) ile sürekli olarak paylaşması gerekmektedir. Her bir konumun mahremiyeti LDP ile korunuyor olsa da, kullanıcının ardışık konumları arasındaki korelasyonlar, kullanıcının gerçek konumlarını tahmin etmek için kullanılabilir.

Bu fikirden yola çıkarak, bu tezde LDP altında sürekli konum paylaşımına karşı yeni çıkarım saldırıları öneriyoruz. Önerdiğimiz saldırılar iki kategoriye ayrılmıştır: sabit kullanıcılar için Bayesçi saldırgan formülasyonuna dayalı istatistiksel saldırılar ve mobil kullanıcılar için Gizli Markov Modeli (HMM) tabanlı saldırılar. Ayrıca, HMM tabanlı saldırılarımız için iki ilerletme öneriyoruz (bilgilendirilmiş ve zincir saldırılar); bu ilerletmeler saldırganın topluluk istatistiklerini saldırıya dahil etmesini veya HMM inşasını birden fazla kez tekrarlamasını sağlayarak saldırı etkinliğini arttırmaktadır. Saldırılarımızı dört popüler LDP protokolüne (GRR, RAPPOR, OUE, OLH), üç veri kümesine ve farklı mahremiyet seviyelerine uyarlıyor ve uyguluyoruz. Deney sonuçları, saldırılarımızın oldukça etkili olduğunu ve sürekli konum paylaşımında LDP kullanımının getirdiği mahremiyet risklerini ortaya koymaktadır. Ayrıca, sonuçlar saldırılarımızın tasarlandıkları kullanıcı kategorileri için daha uygun olduğunu göstermektedir; yani, sabit kullanıcılar için istatistiksel saldırılar HMM tabanlı saldırılara göre daha yüksek etkinlik sağlarken, mobil kullanıcılar için HMM tabanlı saldırılar daha yüksek etkinlik sağlamaktadır.

HMM tabanlı saldırılarımız bitvektör-bazlı LDP protokolleri (örneğin RAPPOR

ve OUE) üzerinde etkili olsa bile, yüksek bellek kullanımı ve uzun yürütme süresi gibi verimlilik sorunları yaşadıklarını gözlemledik. Bu sorunlar, özellikle kafes boyutları büyük olduğunda veya saldırgan hassas konum çıkarımı yapmayı hedeflediğinde saldırılarımızın ölçeklenebilirliğini kısıtlamaktadır. Bu nedenle, saldırı HMM'lerinden düşük gerçekleşme olasılığına sahip durumları çıkartarak saldırılarımızın zaman ve bellek verimliliğini arttıracak buluşsal yöntemler öneriyoruz. Deney sonuçları ile gösteriyoruz ki, yöntemlerimiz sayesinde saldırılar orjinal versiyonlarına kıyasla benzer etkinlik seviyelerini koruyabilirken, bellek ve zaman maliyetleri on kattan fazla azalabilmektedir.

Son olarak, saldırılarımıza karşı üç savunma stratejisi öneriyoruz: Memoization, Replay ve Replication. Memoization yönteminde, her kullanıcı orjinal değerinin bozulmuş bir versiyonunu saklar ve bu değer, veri toplayıcıya raporlanmadan önce ikinci kez bozulur. Replay yönteminde, her kullanıcı önceki konumlarını ve bozulmuş karşılıklarını bir önbellekte saklar. Herhangi bir konum tekrarlandığında, kullanıcı yeni bir bozulma işlemi yapmak yerine eski bozulmuş versiyonu önbellekten çekerek gönderir. Replication yönteminde ise, kullanıcının güncel konumu bir önceki konumuna benzer ise, kullanıcı bir önceki raporladığı değeri tekrarlar. Bu üç savunmayı dört LDP protokolü, üç veri kümesi, farklı mahremiyet seviyeleri ve sabit ve mobil kullanıcıları içeren deney düzeneğimizde değerlendiriyoruz. Deney sonuçları, savunmalarımızın saldırı etkinliğini azaltmada başarılı olduğunu göstermektedir. Ayrıca, savunmaların etkinlik, verimlilik ve yararlılık yönlerini farklı koşullar altında eleştirel bir şekilde analiz ediyor ve hangi durumda hangi savunmanın tercih edilmesi gerektiğine dair öneriler sunuyoruz.

ACKNOWLEDGMENTS

I would like to express my deepest gratitude to my advisor, Asst. Prof. Emre Gürsoy, for his invaluable guidance and unwavering support, which made this work possible. His advice and mentorship were instrumental in navigating every stage of this thesis.

I also extend my heartfelt thanks to the members of my thesis committee, Prof. Alptekin Küpçü and Prof. Yücel Saygın, for their insightful comments and suggestions and for making my defense an enjoyable and enriching experience.

I am profoundly grateful to my family for their steadfast support and understanding throughout the course of my research and writing. Their prayers, encouragement, and unwavering belief in me have been a constant source of strength.

I wish to express my sincere gratitude to my SPADE lab members for their invaluable support and encouragement during my research journey. In particular, I am deeply thankful to Raul Aliyev, Alireza Khodaie, Berkay Kemal Balioğlu, and Emirhan Delen for their insights, constructive feedback, and unwavering collaboration. Their willingness to share knowledge and provide guidance during challenging moments was instrumental in overcoming obstacles and achieving my goals. This thesis would not have been possible without their collective efforts and camaraderie, for which I am profoundly grateful. I would also like to extend my special thanks to Abdullah Saydemir for his invaluable assistance and motivation, which played a significant role in the completion of this thesis.

Finally, this research was supported by The Scientific and Technological Research Council of Türkiye (TUBİTAK) under project number 121E303. I sincerely thank TUBİTAK for their support.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

ABBREVIATIONS

LDP	Local Differential Privacy
GRR	Generalized Randomized Response
OUE	Optimized Unary Encoding
OLH	Optimized Local Hashing
HMM	Hidden Markov Model
PA	Prediction Accuracy
NDE	Normalized Distance Error

Chapter 1

INTRODUCTION

Local differential privacy (LDP) has recently emerged as an accepted privacy standard and received significant attention from the academia and industry [Cormode et al., 2018, Gursoy et al., 2022, Yang et al., 2020]. It has also been deployed in products of various tech companies such as Apple, Google, and Microsoft [app, 2020, Ding et al., 2017, Erlingsson et al., 2014]. With the growing popularity of LDP, several recent works have studied the application of LDP to location data and location-based services (LBS). In particular, LDP has been used for the collection of aggregate statistics (such as density statistics and mobility models) from users' location data [Alptekin and Gursoy, 2023, Chen et al., 2016, Cunningham et al., 2021, Du et al., 2023], as well as to enable privacy-preserving usage of LBS [Hong et al., 2021, Hong et al., 2022, Wang et al., 2022].

On the other hand, in many practical applications of location data sharing and LBS usage, the user must share his/her location continuously with the data collector (e.g., LBS provider). In this case, although LDP perturbs each location to protect its privacy, the data collector observes not a single perturbed location from the user, but rather multiple consecutive perturbed locations. Since consecutive locations of a user are naturally correlated, the perturbed locations observed by the data collector are also correlated. Such correlations can be exploited by an honest-but-curious data collector to infer the user's true locations.

Following this idea, in this thesis we propose novel location inference attacks against continuous location sharing under LDP. Our attacks enable a data collector, who has observed multiple perturbed location readings from a user, to accurately predict the user's true locations. We develop attacks targeting two scenarios: (i)

The user is stationary or near-stationary, e.g., at home, work, or school. In this case, we propose statistical attacks that utilize a Bayesian attack formulation. (ii) The user is mobile, e.g., traveling by car, taxi, bus, etc. In this case, we propose Hidden Markov Model (HMM)-based attacks in which LDP protocol outputs are treated as observable states of an HMM and the user's true locations are treated as hidden states. The Viterbi algorithm [Forney, 1973] is used to infer the most likely sequence of hidden states from the observed states.

We adapt and apply our attacks to four popular LDP protocols: GRR, RAPPOR, OUE, and OLH. Using three datasets, two metrics, four LDP protocols, and various ε, we experimentally show that our attacks are indeed effective. For example, under commonly used ε values such as $\varepsilon = 2$, our statistical attacks on stationary users can predict the users' true locations with accuracy higher than 80% in most cases. We also experimentally show that both attacks are better suited in the scenario that they were designed for, i.e., statistical attacks designed for stationary scenarios achieve higher effectiveness on stationary users, whereas HMM-based attacks designed for mobile scenarios achieve higher effectiveness on mobile users.

We then propose two extensions for our HMM-based attacks: "informed" and "chain" attacks. In the informed attack, the attacker utilizes knowledge of aggregate density and mobility statistics concerning the general user population to improve attack effectiveness. According to our results, an informed attacker has roughly twice as high prediction accuracy compared to an uninformed attacker; in addition, prediction accuracies reach close to 100% when $\varepsilon \geq 2.5$. In the chain attack, we execute the HMM-based attack for multiple iterations, such that the prediction results of each iteration determine the transition probabilities and initial probability distributions of the HMM in the next iteration. This strategy enables us to iteratively improve our HMMs, thereby achieving more accurate attacks especially for high ε.

On the other hand, we observe that our HMM-based attacks suffer from two inefficiency problems when applied to bitvector-based protocols such as RAPPOR and OUE: high memory usage and high execution times. We identify that these problems are caused by the exponential growth in two HMM components: the set of observable states and the emission probability matrix. To address the aforementioned ineffi-

ciency problems, we propose a heuristic method in which those observable states in the HMM with low occurrence probabilities are removed. By carefully determining the number of removed states, we experimentally show that the memory usage of the attack can be reduced by approximately 20 folds and the execution time can be reduced by approximately 35 folds. Furthermore, we show that these reductions are achieved with negligible negative impact on the prediction accuracies of our attacks.

Lastly, we propose three defenses against our attacks: Memoization, Replay, and Replication. The concept of memoization was originally introduced in [Erlingsson et al., 2014] and later used in works such as [Arcolezi et al., 2022] to improve LDP protocols for longitudinal and multidimensional frequency estimation. In our work, we adapt and apply the memoization idea to multiple LDP protocols in the context of continuous location sharing. In the newly proposed replay defense, a private key-value store called the *cache* is introduced to store the user's previous locations and the corresponding perturbed counterparts (with LDP) on the user's device. When a certain location is repeated, the perturbed counterpart is retrieved from the cache and replayed to the data collector. The third defense, replication, aims to circumvent the increasing cache size problem of replay. In replication, the user only stores the perturbed counterpart of the previous location and reports it to the data collector if the location is repeated.

We experimentally evaluate the three defenses using the same experiment setup as before. We explore the effectiveness of the defenses in terms of the reduction in attackers' prediction accuracy (PA) as well as the defenses' efficiency implications. Results show that the defenses are beneficial to defend against the attacks, as they generally reduce attackers' PAs for both stationary and mobile users. Considering the PA reduction and efficiency trade-offs, we find that different defenses can be preferable under different factors and conditions (e.g., different trajectory length, value of ε, stationary or mobile users). We provide recommendations regarding when to use which defense.

1.1 Contributions

In summary, in this thesis we make the following contributions:

- We propose novel inference attacks against continuous location sharing under Local Differential Privacy (LDP). Our attacks are categorized into two types: statistical attacks that focus on stationary users, and Hidden Markov Model (HMM)-based attacks that target mobile users.

- To demonstrate the effectiveness of our attacks, we evaluate them using four LDP protocols, three datasets, two evaluation metrics, and varying ε privacy levels. Our findings underscore the privacy risks associated with continuous location sharing under LDP.

- We extend our HMM-based attack by introducing two variants: informed and chain attacks. Experiment results indicate that these extensions enhance the attacker's ability to achieve higher prediction accuracy.

- We propose heuristic methods to improve the time and memory-efficiency of our attacks on bitvector-based LDP protocols. We show that our improvements can substantially reduce the memory and time costs of our attacks without a significant impact on their effectiveness.

- Finally, we propose three defense strategies against our attacks: Memoization, Replay, and Replication. We experimentally demonstrate the effectiveness of our defenses and discuss their utility and efficiency implications.

1.2 Thesis Organization

The organization of the remainder of this thesis is as follows. In Chapter 2, we present and discuss related work. In Chapter 3, we provide the background and preliminaries related to location data, Local Differential Privacy (LDP), LDP protocols, and Hidden Markov Models (HMMs). Chapter 4 introduces and experimentally evaluates our proposed inference attacks, including both statistical and HMM-based

attacks, along with their informed and chain extensions. Chapter 5 describes our methods for enhancing the time and memory-efficiency of our HMM-based attacks on bitvector-based protocols, along with the corresponding experiment results. Chapter 6 explains the defenses we propose against our inference attacks, experimentally evaluates their effectiveness, and discusses their strengths and weaknesses. Finally, Chapter 7 concludes the thesis.

Chapter 2

RELATED WORK

2.1 Applications of LDP to Location Data

In recent years, the popularity of LDP has sparked interest in its application to location data and LBS. Chen et al. [Chen et al., 2016] proposed methods for spatial data aggregation and density estimation under personalized LDP. Hong et al. [Hong et al., 2021, Hong et al., 2022] studied the problem of reducing the expected error of each perturbed location when collecting location data under LDP. Wang et al. [Wang et al., 2022] developed L-SRR, an LDP version of Staircase Randomized Response (SRR). They applied L-SRR to various LBS tasks, including traffic density estimation and k-nearest neighbor. Yang et al. [Yang et al., 2022] studied the problem of collecting location trajectories under LDP, and Cunningham et al. [Cunningham et al., 2021] proposed a mechanism for trajectory sharing under LDP which utilizes hierarchical n-grams. Du et al. [Du et al., 2023] proposed LDPTrace for synthesizing realistic location trajectories using aggregate statistics collected with LDP. Kim et al. [Kim et al., 2018] and Navidan et al. [Navidan et al., 2022] applied LDP to indoor positioning data. Alptekin and Gursoy [Alptekin and Gursoy, 2023] studied the problem of building quadtrees for locally private spatial decomposition. Tire and Gursoy [Tire and Gursoy, 2024] developed a solution for answering spatial density queries under LDP. Finally, Balioglu et al. [Balioglu et al., 2024] studied grid-based decomposition methods for location data, and proposed a new method for advancing adaptive grids in LDP. Considering these recent works on applying LDP to location data and LBS, it becomes important to highlight the potential privacy leakages of continuous location sharing with LDP, which is the goal of our work.

2.2 Privacy Attacks on LDP

The popularity of LDP has also fueled the development of novel privacy attacks against LDP protocols and implementations. Murakami et al. [Murakami and Takahashi, 2021] proposed a metric to measure re-identification risk under LDP by linking perturbed data to users. Gursoy et al. [Gursoy et al., 2022] studied the privacy protection of LDP protocols from a Bayesian perspective. Gadotti et al. [Gadotti et al., 2022] proposed pool inference attacks in which the attacker defines pools of interest, and then aims to infer the user's preferred pool. The attack is evaluated with Apple's Count Mean Sketch (CMS) algorithm. Arcolezi et al. [Arcolezi et al., 2023] studied the privacy risks that arise from collecting multidimensional records under LDP. Gursoy [Gursoy, 2024] performed longitudinal attacks on iterative data collection using LDP to illustrate how collecting multiple responses from the same user over time can degrade that user's privacy.

While these works propose interesting attacks, unlike us, they do not study location data. Our work advances the existing line of research on privacy attacks and evaluations on LDP, particularly for LDP applications to location data, which have not been studied before.

2.3 Poisoning Attacks on LDP

Another popular category of attacks on LDP is *poisoning attacks* in which malicious users poison their LDP protocol inputs or outputs to manipulate the statistics recovered on the server side. The susceptibility of LDP protocols to poisoning attacks was initially shown by Cheu et al. [Cheu et al., 2021] and Cao et al. [Cao et al., 2021]. Their attack methods were demonstrated using the frequency estimation task and established LDP protocols such as GRR, OUE, and OLH. Further studies by Li et al. [Li et al., 2023] introduced poisoning attacks aimed at calculating mean and variance. Meanwhile, Li et al. [Li et al., 2024] examined how resilient LDP protocols tailored for numerical attributes are against poisoning attacks. Wu et al. [Wu et al., 2022] introduced poisoning strategies for LDP protocols geared towards key-value data collection, and Imola et al. [Imola et al., 2022] explored the robustness of LDP

protocols against poisoning attacks in the context of graph analysis. Lastly, Zheng et al. [Zheng et al., 2024] introduced optimal data poisoning attacks and corresponding defense mechanisms for LDP-based privacy-preserving crowdsensing.

Recent literature has also discussed various defenses against poisoning attacks. Kato et al. [Kato et al., 2021] developed secure, efficient, and verifiable LDP protocols to prevent manipulation and poisoning attacks. They employed the Cryptographic Randomized Response Technique (CRRT) as a key component to adapt existing LDP mechanisms into verifiable versions. Song et al. [Song et al., 2023] presented two novel LDP protocols where both the aggregator and client are involved in the perturbation process. This dual involvement enables users to self-verify and ensure that they are not participating in a poisoning attack. Huang et al. [Huang et al., 2024] addressed the issue of defending against data poisoning attacks in LDP protocols by introducing LDPGuard, a framework that uses a two-round data collection method to estimate the percentage of malicious users and develop strategies to counter these attacks.

The attacks proposed in this thesis do not perform any poisoning or client-side manipulation. Instead of assuming malicious clients as adversaries (which is the case in poisoning attacks), we assume that the adversary is an honest-but-curious server or unauthorized third party. Hence, previous works falling under the poisoning attack category are not comparable to the attacks we present in this thesis.

Chapter 3

PRELIMINARIES

3.1 Location Data and LDP

Consider a two-dimensional geolocation space $\mathcal{L}$ and user population $\mathcal{P}$. For user $u \in \mathcal{P}$, the user's true location at time t is denoted by l_u^t, where $l_u^t \in \mathcal{L}$ is a tuple consisting of latitude and longitude coordinates. In recent years, several works have studied the application of local differential privacy (LDP) to enable privacy-preserving sharing of l_u^t with a data collector or LBS provider [Alptekin and Gursoy, 2023, Hong et al., 2021, Wang et al., 2022, Hong et al., 2022, Navidan et al., 2022]. In this context, LDP is defined as follows.

Definition 1 *A randomized algorithm Ψ satisfies ε-local differential privacy (ε-LDP), where $\varepsilon > 0$, if and only if for any two locations l_i, l_i^*, it holds that:*

$$\forall y \in Range(\Psi): \quad \frac{Pr[\Psi(l_i) = y]}{Pr[\Psi(l_i^*) = y]} \leq e^{\varepsilon} \tag{3.1}$$

where $Range(\Psi)$ stands for the set of all possible outputs of the algorithm Ψ.

ε-LDP ensures that a party observing the output y (e.g., the data collector) will not be able to distinguish between the user's actual location l_i and an incorrect location l_i^* beyond the ratio e^{ε}. The amount of privacy is controlled by the ε parameter. Lower ε results in increased privacy.

When each user applies $\Psi(l_u^t)$ on their location and then shares the output with the data collector, the privacy of l_u^t remains protected with LDP. On the other hand, in many practical applications of LBS usage and location sharing, the user shares his/her location *continuously* with the data collector, i.e., the data collector observes a sequence of outputs from u at consecutive timestamps: $\mathcal{O}_u = \{\Psi(l_u^1), \Psi(l_u^2), ..., \Psi(l_u^n)\}$, where $1 \leq t \leq n$ denote the timestamps. Considering that

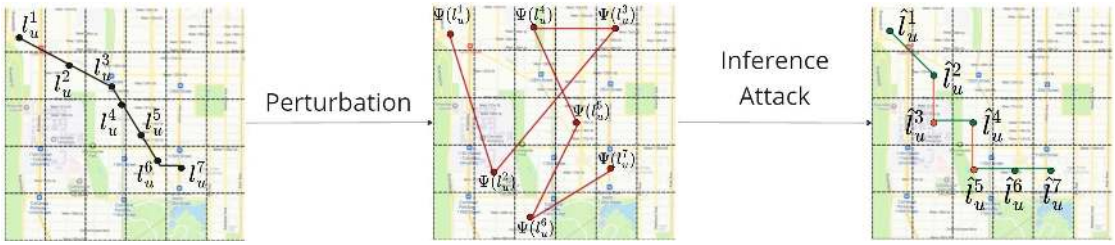

Figure 3.1: A visual overview of our attack scenario. On the left, user u with locations $l_u^1, l_u^2, ..., l_u^7$ is traveling in the city. The user shares his/her locations with a data collector (e.g., LBS provider) after perturbing each location with LDP. The data collector observes the trajectory in the middle. Then, using our inference attacks, the data collector is able to predict the user's locations (on the right).

consecutive locations of a user (e.g., l_u^t, l_u^{t+1}) are naturally correlated in the physical world, the intuition behind our attacks is to exploit these relationships to infer the user's true locations from the set of LDP outputs $\mathcal{O}_u$. A sample workflow of our attack is illustrated in Figure 3.1: actual locations $l_u^1, l_u^2, ..., l_u^7$ on the left, attacker's observed $\mathcal{O}_u = \{\Psi(l_u^1), \Psi(l_u^2), ..., \Psi(l_u^7)\}$ in the middle, and results of our inference attack (predicted locations) on the right.

3.2 LDP Protocols

The popularity of the LDP definition has led to the development of several LDP protocols [Cormode et al., 2018, Gursoy et al., 2022, Wang et al., 2017, Xiong et al., 2020]. New systems also typically use these LDP protocols as building blocks. In this thesis, in order to demonstrate the effectiveness of our attacks on multiple LDP protocols, we choose the following four popular and commonly used protocols from the literature: GRR, RAPPOR, OUE, and OLH [Warner, 1965, Erlingsson et al., 2014, Kairouz et al., 2016, Wang et al., 2017]. Since these protocols require discrete inputs, we discretize the geolocation space $\mathcal{L}$ using a $\beta \times \gamma$ grid $\mathcal{G}$, which consists of cells C_1, C_2, ..., $C_{\beta\times\gamma}$. A sample 5×6 grid is shown in Figure 3.1. Then, each user discretizes their current location l_u^t according to which cell their location falls into. This cell becomes the true value v that the user feeds into the LDP protocol.

The output of the LDP protocol is shared with the data collector. Below, denoting by v the user's input to the LDP protocol and by $|\mathcal{G}|$ the number of cells in $\mathcal{G}$, we describe each protocol briefly.

3.2.1 Generalized Randomized Response (GRR)

GRR is built on top of the randomized response survey collection technique initially proposed in [Warner, 1965]. Given the user's true value v as input, it decides which value v^* is reported to the data collector as follows: $v^* \leftarrow v$ with probability p, and with probability $1-p$, a fake item is sampled uniformly at random from $\mathcal{G}$ and reported to the data collector. This process satisfies ε-LDP when $p = \frac{e^\varepsilon}{e^\varepsilon + |\mathcal{G}| - 1}$. In short, the perturbation algorithm Ψ_{GRR} performs the following:

$$\Pr[\Psi_{\text{GRR}}(v) = y] = \begin{cases} p = \frac{e^\varepsilon}{e^\varepsilon + |\mathcal{G}| - 1} & \text{if } y = v \\ q = \frac{1}{e^\varepsilon + |\mathcal{G}| - 1} & \text{if } y \neq v \end{cases} \tag{3.2}$$

3.2.2 RAPPOR

In RAPPOR, the user's true value v is encoded into a bitvector and then the bitvector is perturbed in a randomized way to satisfy LDP. While the original version of RAPPOR utilizes Bloom filters for encoding strings [Erlingsson et al., 2014, Fanti et al., 2016], here we use a version of RAPPOR that utilizes one-hot encoding [Wang et al., 2017, Gursoy et al., 2022].

Given the user's true value v, first a bitvector B is initialized with length equal to $|\mathcal{G}|$. In B, only the position corresponding to v contains a 1 bit: $B[v] = 1$. All remaining positions contain 0 bits: $\forall i \neq v : \quad B[i] = 0$. Then, the perturbation algorithm Ψ_{RAPPOR} takes B as input and produces a perturbed bitvector B'. This perturbed bitvector is constructed by iterating through each bit in B one by one, and either keeping or flipping that bit with probabilities controlled by ε:

$$\forall i \in [1, |\mathcal{G}|] : \quad \Pr[B'[i] = 1] = \begin{cases} \frac{e^{\varepsilon/2}}{e^{\varepsilon/2} + 1} & \text{if } B[i] = 1 \\ \frac{1}{e^{\varepsilon/2} + 1} & \text{if } B[i] = 0 \end{cases} \tag{3.3}$$

3.2.3 Optimized Unary Encoding (OUE)

OUE's encoding step is the same as RAPPOR, i.e., bitvector B is initialized with length $|\mathcal{G}|$, only a single position $B[v] = 1$, and all remaining positions contain $B[i] = 0$ for $i \neq v$. However, OUE's perturbation algorithm Ψ_{OUE} is different because it treats the 0 and 1 bits asymmetrically, i.e., the probabilities of keeping and flipping the 0 bits are different from those of 1 bits. More concretely, in case of OUE, the user's perturbed bitvector B' is constructed as follows:

$$\forall i \in [1, |\mathcal{G}|]: \quad \Pr[B'[i] = 1] = \begin{cases} \frac{1}{2} & \text{if } B[i] = 1 \\ \frac{1}{e^{\varepsilon}+1} & \text{if } B[i] = 0 \end{cases} \tag{3.4}$$

3.2.4 Optimized Local Hashing (OLH)

OLH utilizes hash functions to encode the user's data and then performs GRR in the hashed domain. Let $\mathcal{H}$ be a universal hash function family where each $H \in \mathcal{H}$ maps a value from $\mathcal{G}$ into an integer in $[1, k]$, i.e., $H : \mathcal{G} \to [1, k]$. First, each user draws a hash function uniformly at random from $\mathcal{H}$, i.e.: $H_u \leftarrow_{\$} \mathcal{H}$. Then, the user computes integer x by hashing their v, i.e.: $x = H_u(v)$. Then, the perturbation algorithm Ψ_{OLH} takes as input x and outputs perturbed integer $x' \in [1, k]$ with the probabilities:

$$\Pr[\Psi_{\text{OLH}}(x) = x'] = \begin{cases} \frac{e^{\varepsilon}}{e^{\varepsilon}+k-1} & \text{if } x = x' \\ \frac{1}{e^{\varepsilon}+k-1} & \text{if } x \neq x' \end{cases} \tag{3.5}$$

The user sends $\langle H_u, x' \rangle$ to the data collector. A key parameter of the OLH protocol is k. In the rest of this thesis, we use the default value $k = e^{\varepsilon} + 1$ as derived in [Wang et al., 2017] and used in follow-up works.

3.3 Hidden Markov Models (HMMs)

Our attacks utilize Hidden Markov Models (HMMs). A HMM is a statistical model that follows the Markov property, i.e., the transition at each step depends only on the previous transition and not the full transition history [Baum and Petrie, 1966, Rabiner and Juang, 1986]. In a HMM, there is an observable process whose outcomes

depend on the outcomes of an unobservable ("hidden") underlying stochastic process. Since the underlying process cannot be observed, the goal of HMM modeling is to learn more about the state of the underlying process through the observations made from the observable process. A HMM consists of the following components:

- A set of observable states $O = \{o_1, o_2, ..., o_N\}$.
- A set of hidden states $S = \{s_1, s_2, ..., s_K\}$.
- A matrix of transition probabilities $\mathcal{A}$ with size $K \times K$. In this matrix, $\mathcal{A}[i, j]$ represents the transition probability from hidden state s_i to s_j, i.e., given the current state $= s_i$, what is the probability that the next state is s_j?
- A matrix of emission probabilities $\mathcal{B}$ with size $K \times N$. In this matrix, $\mathcal{B}[i, j]$ represents the emission probability of observed state o_j given that the current hidden state is s_i.
- The initial probability distribution $\Pi = (\pi_1, \pi_2, ..., \pi_K)$. Here, π_i denotes the probability that the initial (very first) state of the hidden process starts at state s_i.

Viterbi algorithm. Consider that a sequence of observations $Y = y_1 y_2 ... y_T$ has been made (for each observation, $y_i \in O$). The Viterbi algorithm is a dynamic programming algorithm for finding the most likely sequence of hidden states that caused Y to occur. It is widely used in HMM-related tasks in fields such as bioinformatics, speech recognition, and computational linguistics [Forney, 1973, Rabiner, 1989, Viterbi, 1967]. A formal description of the Viterbi algorithm is presented in Algorithm 1. The algorithm takes the HMM components O, S, $\mathcal{A}$, $\mathcal{B}$, Π as well as the sequence of observations Y as input. It starts by initializing the probabilities of each state and then iterates through each observation $y_2 ... y_T$, calculating the probabilities of being in each state based on the previous time step's probabilities and the transition probabilities between states. The state probabilities are updated in each step and the most likely state sequence is tracked using backpointers. Once the

Algorithm 1 Viterbi Algorithm

Input: Set of observable states O, set of hidden states S, transition probability matrix $\mathcal{A}$, emission probability matrix $\mathcal{B}$, initial probability distribution Π, sequence of observations Y

Output: Most probable hidden state sequence X

Let $T \leftarrow |Y|$
$Probs \leftarrow$ Matrix of size $K \times T$ initialized to zero
$Backpointers \leftarrow$ Matrix of size $K \times T$ initialized to zero
for $i = 1$ to K **do**
    $Probs[i, 1] \leftarrow \pi_i \cdot \mathcal{B}[i, y_1]$
    $Backpointers[i, 1] \leftarrow 0$
end for
for $j = 2$ to T **do**
    for $i = 1$ to K **do**
        $Probs[i, j] \leftarrow \max_k (Probs[k, j-1] \cdot \mathcal{A}[k, i] \cdot \mathcal{B}[i, y_j])$
        $Backpointers[i, j] \leftarrow \operatorname{argmax}_k (Probs[k, j-1] \cdot \mathcal{A}[k, i] \cdot \mathcal{B}[i, y_j])$
    end for
end for
Let $X = x_1 x_2 \dots x_T$ be the best path, initialized as $x_i = \emptyset$ for all $i \in [1, T]$
$z_T \leftarrow \operatorname{argmax}_k(Probs[k, T])$
$x_T \leftarrow S[z_T]$
for $j = T$ to 2 **do**
    $z_{j-1} \leftarrow Backpointers[z_j, j]$
    $x_{j-1} \leftarrow S[z_{j-1}]$
end for
return X

algorithm reaches y_T, it identifies the best state sequence by backtracking through the stored backpointers.

Chapter 4

OUR INFERENCE ATTACKS

For user u, recall that $l_u^1, l_u^2, ..., l_u^n$ denote the user's true locations, and $\mathcal{O}_u = \{\Psi(l_u^1), \Psi(l_u^2), ..., \Psi(l_u^n)\}$ denote the observations made by the attacker. The attacker can be the data collector, honest-but-curious LBS provider, man-in-the-middle between user u and the data collector, or any party who observes $\mathcal{O}_u$. Ψ denotes an LDP protocol (e.g., one of GRR, RAPPOR, OUE, OLH) used to protect the privacy of the user's locations. Armed with $\mathcal{O}_u$, the goal of the attacker is to infer $l_u^1, l_u^2, ..., l_u^n$. We propose attacks for two scenarios:

- The user is stationary or near-stationary. This scenario is relevant when the user is at home, work, school, etc. In this case, since the user's locations $l_u^1, l_u^2, ..., l_u^n$ are identical or very similar, we propose to leverage statistical attacks formulated using a Bayesian attacker model.

- The user is mobile, i.e., in transit. This scenario is relevant when the user is actively traveling, e.g., by car or bus. In this case, there are noticeable differences between consecutive l_u^t and l_u^{t+1}; however, the two are still correlated. We propose to leverage HMM-based attacks in this more challenging case.

4.1 Statistical Attacks Against Stationary Users

In this scenario, the user's locations are $l_u^1, \approx l_u^2 \approx ... \approx l_u^n$, therefore let l_u denote a representative location (e.g., the mean or median of $l_u^1, l_u^2, ..., l_u^n$). Given $\mathcal{O}_u$, the goal of the attacker is to predict l_u. Denoting by $\hat{l}_u$ the attacker's prediction, the

Bayes-optimal attack strategy is to select $\hat{l}_u$ as:

$$\hat{l}_u = \underset{l \in \mathcal{L}}{\text{argmax}} \ \Pr[l|\mathcal{O}_u] = \underset{l \in \mathcal{L}}{\text{argmax}} \ \frac{\Pr[\mathcal{O}_u|l] \cdot \Pr[l]}{\Pr[\mathcal{O}_u]} = \underset{l \in \mathcal{L}}{\text{argmax}} \ \Pr[\mathcal{O}_u|l] \cdot \Pr[l]$$
$$\propto \underset{l \in \mathcal{L}}{\text{argmax}} \ \Pr[\mathcal{O}_u|l]$$

assuming a uniform prior, i.e., uniform $\Pr[l]$ across all $l \in \mathcal{L}$. (Non-uniform priors are explored in Section 4.3.) Then, it is necessary to compute $\Pr[\mathcal{O}_u|l]$ for different locations l, so that we can find which l has the highest likelihood of causing $\mathcal{O}_u$. The specifics of this computation are protocol-dependent since each protocol outputs a different data structure, e.g., RAPPOR and OUE report bitvectors whereas OLH reports a tuple $\langle H_u, x' \rangle$. Below, we describe the application of our attack strategy to each of the different protocols under consideration.

4.1.1 Attacking GRR

In GRR, the user sends a cell C from $\mathcal{G}$ to the data collector. Thus, the observations are: $\mathcal{O}_u = \{C_u^1, C_u^2, ..., C_u^n\}$ such that each $C_u^i \in \mathcal{G}$. We observe from Equation 3.2 that the probability p of sending the true cell of the user is higher than the probability q of sending any fake cell. Hence, the Bayes-optimal prediction strategy is to find the cell in $\mathcal{O}_u$ with the highest number of occurrences, i.e., find the mode of $\mathcal{O}_u$. Let $\text{count}(\mathcal{O}_u, C_i)$ be a function that counts the number of occurrences of cell C_i in $\mathcal{O}_u$; then, $\hat{l}_u$ is selected as:

$$\hat{l}_u = \underset{C_i \in \mathcal{G}}{\text{argmax}} \ \text{count}(\mathcal{O}_u, C_i) \tag{4.1}$$

4.1.2 Attacking RAPPOR and OUE

In RAPPOR and OUE, the user sends bitvectors of length $|\mathcal{G}|$ to the data collector. Thus, the observations are $\mathcal{O}_u = \{B_u^1, B_u^2, ..., B_u^n\}$. In both protocols, the decision to keep or flip each bit is independent of the other bits. For RAPPOR, we observe from Equation 3.3 that the probability to keep each bit is $\frac{e^{\varepsilon/2}}{e^{\varepsilon/2}+1}$ whereas the probability to flip each bit is $\frac{1}{e^{\varepsilon/2}+1}$. For OUE, from Equation 3.4 the probability of flipping an original 0 bit is $\frac{1}{e^{\varepsilon}+1}$ whereas the probability of keeping it is high. In contrast, the probability of keeping or flipping an original 1 bit is $\frac{1}{2}$. Thus, in both protocols,

when a 1 bit is observed in an index of a reported bitvector B_u^i, that index has higher likelihood of originally containing a 1 bit compared to an index that contains a 0 bit in B_u^i. Hence, after obtaining $\mathcal{O}_u$, the attacker computes a *sum* vector of length $|\mathcal{G}|$ such that:

$$\text{sum}[j] = \sum_{i=1}^{n} B_u^i[j] \tag{4.2}$$

That is, the reported bitvectors $B_u^1, B_u^2, ..., B_u^n$ are summed index by index. Then, the index with the highest sum value is selected:

$$\hat{l}_u = \underset{j \in [1,|\mathcal{G}|]}{\text{argmax}} \; sum[j] \tag{4.3}$$

4.1.3 Attacking OLH

In OLH, the user sends tuples containing hash functions (drawn from a hash family) and integers in range $[1, k]$ to the data collector. Thus, the observations are: $\mathcal{O}_u = \{\langle H_u^1, x_u^1\rangle, \langle H_u^2, x_u^2\rangle, ..., \langle H_u^n, x_u^n\rangle\}$. Recall from Equation 3.5 that OLH applies GRR in the hashed domain $[1, k]$, therefore the probability of sending the true integer $x_u = H_u(v)$ is higher than any other integer in $[1, k]$. As a result, the Bayes-optimal prediction strategy is similar to GRR. However, since there is an additional hashing component in OLH, the attacker must compute which values hash to x_u (there can be multiple) and find the highest number of occurrences afterward.

Let $\text{support}(H_u, x_u)$ be defined as the set of cells in $\mathcal{G}$ whose outcomes are equal to x_u when hashed with hash function H_u, i.e.:

$$\text{support}(H_u, x_u) = \{C_i \in \mathcal{G} \mid H_u(C_i) = x_u\} \tag{4.4}$$

After obtaining $\mathcal{O}_u$, the attacker performs the following steps:

1. A count vector of length $|\mathcal{G}|$ is initialized, such that initially, $\text{count}[j] = 0$ for all $j \in [1, |\mathcal{G}|]$.

2. For each tuple $\langle H_u^i, x_u^i\rangle$ in $\mathcal{O}_u$, $\text{support}(H_u, x_u)$ is computed. For each $C_j \in \text{support}(H_u, x_u)$, $\text{count}[j]$ is incremented by $+1$.

3. The index with the highest count is selected: $\hat{l}_u = \underset{j \in [1,|\mathcal{G}|]}{\text{argmax}} \; \text{count}[j]$.

4.2 HMM-Based Attacks Against Mobile Users

In this scenario, the attacker observes $\mathcal{O}_u = \{\Psi(l_u^1), \Psi(l_u^2), ..., \Psi(l_u^n)\}$ from a mobile (moving) user and wants to infer the user's underlying locations $l_u^1, l_u^2, ..., l_u^n$. We formulate this problem as a HMM problem – the LDP protocol outputs are observable but the user's underlying true locations are hidden. Given a true location, the probabilities of an LDP protocol yielding the observed outcomes are treated as the emission probabilities (we derive the emission probabilities from the protocol definitions). Transition probabilities between hidden states are not known by the attacker, but they can be instantiated using realistic assumptions, e.g., given that the user is currently in location l_u^t, the user cannot travel extremely far away in the next timestamp. Thus, the probability that the user is near l_u^t at time $t+1$ is higher than the probability that the user is far away from l_u^t at time $t+1$. Finally, using this formulation, the problem of finding the most likely sequence of locations $l_u^1, l_u^2, ..., l_u^n$ that caused $\mathcal{O}_u$ to occur is solved using the Viterbi algorithm.

Having explained the high-level intuition of our attack above, we now describe the mathematical formulations in more detail. Recall from Section 3.3 that a HMM is characterized using five components: O, S, $\mathcal{A}$, $\mathcal{B}$, and Π. In our attack:

- The set of hidden states S corresponds to cells of the grid $\mathcal{G}$. Each cell $C_i \in \mathcal{G}$ is represented by one hidden state in S.

- The initial probability distribution Π is assigned as a uniform distribution. That is: $\pi_i = \frac{1}{|\mathcal{G}|}$ for all $\pi_i \in \Pi$.

- The transition probability matrix $\mathcal{A}$ contains the probability that a user moves from one cell to another, i.e., $\mathcal{A}[i,j]$ denotes the probability that the user will move from cell C_i to C_j. Denoting by $\text{Adj}(C_i)$ the set of cells adjacent to C_i, we fill the matrix $\mathcal{A}$ as:

$$\mathcal{A}[i,j] = \begin{cases} \frac{1}{|\text{Adj}(C_i)|+1} & \text{if } C_j \in \text{Adj}(i) \text{ or } j = i \\ 0 & \text{otherwise} \end{cases} \tag{4.5}$$

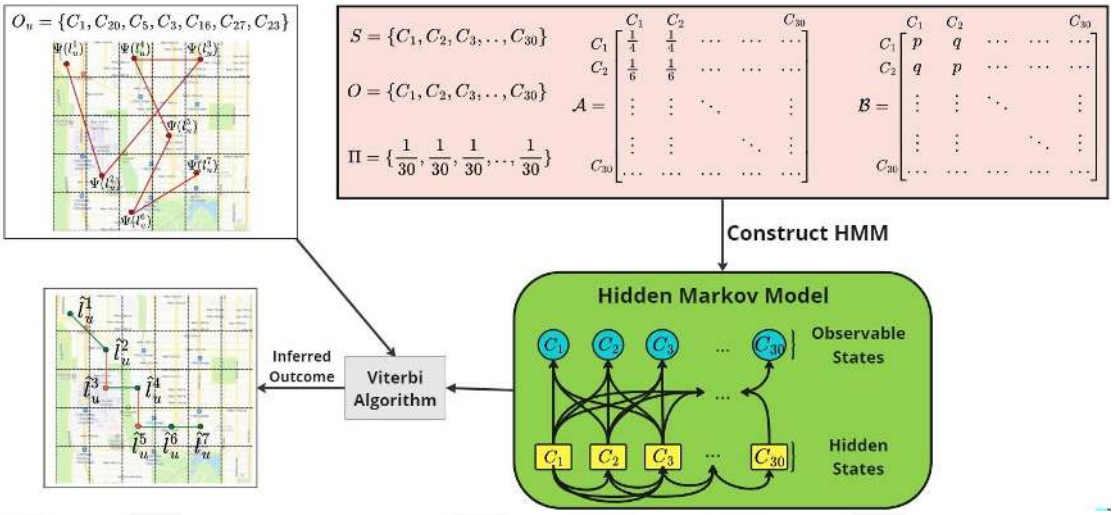

Figure 4.1: Sample illustration of our HMM-based attack for GRR

The rationale behind $\mathcal{A}$ is that in consecutive timestamps, the user may stay in his/her current cell (say C_i) or move to one of the adjacent cells (one of $\text{Adj}(C_i)$). However, the user is unlikely to move a large amount, i.e., to a non-adjacent cell. Therefore, the transition probability from the current cell C_i to non-adjacent cells is set to be 0, whereas the transition probability from C_i to itself (C_i) or any one of the adjacent cells is set following a uniform distribution.

The two remaining components of HMMs are the set of observable states O and the emission probability matrix $\mathcal{B}$. These components change from one LDP protocol to another. Thus, we describe the O and $\mathcal{B}$ for each protocol below.

4.2.1 HMM components for GRR

In GRR, since users send cells from $\mathcal{G}$ to the data collector, the set of observable states O corresponds to the set of grid cells in $\mathcal{G}$. Each cell in $\mathcal{G}$ is represented using one state in O. The emission probability matrix $\mathcal{B}$ is then a $|\mathcal{G}| \times |\mathcal{G}|$ matrix, where:

$$\mathcal{B}[i,j] = \begin{cases} p = \frac{e^{\varepsilon}}{e^{\varepsilon}+|\mathcal{G}|-1} & \text{if } i = j \\ q = \frac{1}{e^{\varepsilon}+|\mathcal{G}|-1} & \text{if } i \neq j \end{cases} \tag{4.6}$$

Our HMM-based attack for GRR is exemplified in Figure 4.1. The user's locations $\mathcal{O}_u$ are observed by the attacker (top left). Since there exist 30 cells in the grid, S consists of $\{C_1, C_2, ..., C_{30}\}$ and Π is filled with probabilities $\frac{1}{30}$. Since GRR protocol was used, O is $\{C_1, C_2, ..., C_{30}\}$. The transition matrix $\mathcal{A}$ is filled for each cell according to how many adjacent cells it has in the grid, e.g., C_1 has 3 adjacent cells plus itself, therefore its probabilities are $\frac{1}{4}$. C_2 has 5 adjacent cells plus itself, therefore its probabilities are $\frac{1}{6}$. The emission matrix $\mathcal{B}$ is a 30x30 matrix, filled with p and q probabilities determined by the GRR protocol (Equation 4.6). Next, a HMM is built using S, O, Π, $\mathcal{A}$ and $\mathcal{B}$. Finally, the HMM and the attacker's observations are fed into the Viterbi algorithm, and the predicted location sequence is obtained, as shown in the bottom left.

4.2.2 HMM components for RAPPOR and OUE

In RAPPOR and OUE, the user sends bitvectors of length $|\mathcal{G}|$ to the data collector. Thus, the set of observable states O corresponds to the bitvectors. For determining $\mathcal{B}[i, j]$, the following steps are performed. Let b_i denote the one-hot encoded version of index i, i.e.: b_i is a length $|\mathcal{G}|$ bitvector full of 0s, but only $b_i[i] = 1$. Let b_j denote the observed bitvector $b_j = O[j]$. Then, $\mathcal{B}[i, j]$ is computed as:

$$\mathcal{B}[i, j] = \prod_{t=1}^{|\mathcal{G}|} \Phi(b_i[t], b_j[t]) \tag{4.7}$$

where $\Phi(b_i[t], b_j[t])$ is computed as follows for RAPPOR:

$$\Phi(b_i[t], b_j[t]) = \begin{cases} \frac{e^{\varepsilon/2}}{e^{\varepsilon/2}+1} & \text{if } b_i[t] = b_j[t] \\ \frac{1}{e^{\varepsilon/2}+1} & \text{if } b_i[t] \neq b_j[t] \end{cases} \tag{4.8}$$

and as follows for OUE:

$$\Phi(b_i[t], b_j[t]) = \begin{cases} \frac{1}{2} & \text{if } b_i[t] = b_j[t] \\ \frac{1}{e^{\varepsilon}+1} & \text{if } b_i[t] \neq b_j[t] \end{cases} \tag{4.9}$$

The reason that the computation of $\Phi(b_i[t], b_j[t])$ is different in RAPPOR and OUE is because they follow from the definitions of the protocols, i.e., Equations 3.3 and 3.4. Since the bit keeping and bit flipping probabilities are different in RAPPOR and OUE, the computation of $\Phi(b_i[t], b_j[t])$ is also different.

4.2.3 HMM components for OLH

In OLH, the user sends hash functions and hash outputs (each output is an integer between $[1, k]$) to the data collector. Thus, the set of observable states O corresponds to integers 1 to k, i.e.: $O = \{1, 2, .., k\}$. The emission probability matrix $\mathcal{B}$ is then a $|\mathcal{G}| \times k$ matrix, where:

$$\mathcal{B}[i, j] = \begin{cases} \frac{e^{\varepsilon}}{e^{\varepsilon}+k-1} & \text{if } H_u(i) = j \\ \frac{1}{e^{\varepsilon}+k-1} & \text{if } H_u(i) \neq j \end{cases} \tag{4.10}$$

The computation of $\mathcal{B}[i, j]$ uses H_u, which is typically different from user to user. We address this by building a different emission matrix for each different user. Note that it is indeed realistic for a real-world attacker to build a different emission matrix for each user, because by definition of the OLH protocol, users share their H_u with the data collector (i.e., H_u is not kept secret). On the other hand, since building a different emission matrix for each user is not necessary for the remaining protocols, the attack for OLH can be slower.

4.3 Extensions

In this section, we describe two extensions for our HMM-based attacks: Informed Attack and Chain Attack.

4.3.1 Informed Attack

Recall from the previous section that the initial probability distribution Π is set as the uniform distribution, and the transition probabilities $\mathcal{A}$ are treated as if the user has equal (uniform) probability of remaining in their current cell and transitioning to each of the adjacent cells. The reason behind these choices is that we assumed a "worst-case" (weak) attacker who does not know any aggregate statistics about the distribution of users or their movement. However, in reality, aggregate population density statistics and aggregate traffic statistics for a given city can be shared by governments of big cities, or they can be observed from navigation apps (e.g.,

Google Maps, Yandex Maps). Thus, the attacker can incorporate such knowledge to strengthen their attack.

The "informed attack" extension stems from this observation. In the informed attack, the attacker is assumed to know the aggregate population density and the transition probabilities between cells for the whole user population $\mathcal{P}$ (but not the locations or transition probabilities of any individual user). The attacker uses these population-level aggregate statistics as his/her Π and $\mathcal{A}$. Using this new Π and $\mathcal{A}$, the attacker is able to build an HMM that better captures the general users' behavior. Consequently, Viterbi decoding also becomes more accurate.

4.3.2 Chain Attack

In our original attack, the problem of finding the most likely sequence of locations $l_u^1, l_u^2, ..., l_u^n$ that caused $\mathcal{O}_u$ to occur is solved using HMM and the Viterbi algorithm once. However, as noted above, this solution uses suboptimal Π and $\mathcal{A}$. The idea behind the chain attack is to use the predicted locations $\hat{l}_u^1, \hat{l}_u^2, ..., \hat{l}_u^n$ to construct Π and $\mathcal{A}$. More concretely, the chain attack performs the following: In the zeroth iteration, an attack is executed with uniform Π and $\mathcal{A}$, identical to the previous section. $\hat{l}_u^1, \hat{l}_u^2, ..., \hat{l}_u^n$ for all users $u \in \mathcal{P}$ are predicted. In the next iteration, another HMM-based attack is executed, but this time, Π and $\mathcal{A}$ are built using the predicted locations $\hat{l}_u^1, \hat{l}_u^2, ..., \hat{l}_u^n$ of users from the previous iteration. New predicted locations are obtained. In the next iteration, another HMM-based attack is executed using Π and $\mathcal{A}$ built using predicted locations from the previous iteration, and so forth.

If the predicted $\hat{l}_u^1, \hat{l}_u^2, ..., \hat{l}_u^n$ are accurate, then Π and $\mathcal{A}$ built from them will be accurate. Since more accurate Π and $\mathcal{A}$ are used in the next iteration, attack results also become more accurate. Thus, the intuition behind the chain attack is that by repeating this process iteratively, we can improve the predictions. Yet, a crucial question is how many iterations to perform. After a few iterations, Π and $\mathcal{A}$ start to converge; therefore, performing more iterations becomes redundant since the results do not change. In addition, since Π and $\mathcal{A}$ are learned from prediction results, any errors in prediction results keep propagating and amplifying in each

iteration. To address this, we introduce a parameter α, which signifies the number of added iterations performed by the chain attack. In Section 4.4, we experimentally demonstrate the impact of varying α on attack effectiveness.

4.4 Experimental Evaluation

4.4.1 Experiment Setup, Datasets, and Metrics

In this section, we experimentally evaluate our statistical and HMM-based attacks. We performed all implementations and experiments in Python. We tested our attacks on three datasets: Taxi, Geolife, and Brinkhoff.

The Taxi dataset was originally released for the Taxi Service Prediction Competition conducted at the ECML-PKDD 2015 conference [Moreira-Matias et al., 2013]. It comprises trajectories of taxis in Porto over one year, with each trajectory representing an individual taxi trip. To eliminate outliers and focus on denser areas of Porto, we pre-processed the dataset by limiting the latitude values between 41.14 (minimum) to 41.18 (maximum) and the longitude values between -8.68 (minimum) to -8.58 (maximum). Following this preprocessing step, our dataset consisted of 1,303,485 distinct taxi trips.

The Geolife dataset comprises GPS trajectory data sourced from the Geolife project at Microsoft Research Asia [Zheng et al., 2010], spanning a period exceeding three years. Trajectories of the users were recorded using various GPS loggers and GPS-enabled smartphones. Each GPS trajectory is represented as a sequence of timestamped location points, including latitude and longitude. Overall, the dataset contains 17,621 trajectories, covering a cumulative distance exceeding 1.2 million kilometers.

The Brinkhoff dataset was simulated using the well-known "Brinkhoff network-based generator of moving objects" [Brinkhoff, 2002]. The motions of 50,000 automobiles were simulated using the map of Oldenburg, Germany. Locations of the automobiles were recorded at equal time intervals.

Recall that the user's true locations are denoted by $l_u^1, l_u^2, ..., l_u^n$ and the locations predicted (inferred) by our attack are denoted by $\hat{l}_u^1, \hat{l}_u^2, ..., \hat{l}_u^n$. We utilize two metrics

Dataset	**Average Trajectory Length**	**Number of Users**
Taxi	37.72	106,659
Geolife	28.64	13,576
Brinkhoff	58.45	18,623

Table 4.1: Summary information regarding the datasets

to quantify the resemblance between them.

Prediction Accuracy (PA). For user $u \in \mathcal{P}$, let $len(u)$ denote the length of the user's trajectory, i.e., number of locations the user shares under LDP. Furthermore, let $gr(l, \mathcal{G})$ be a function that takes a location l and grid $\mathcal{G}$ as input, and returns which cell the location l falls in $\mathcal{G}$. Then, PA is measured as the ratio of cases in which the user's true location and the predicted location fall within the same cell:

$$\text{Prediction Accuracy (PA)} = \frac{\sum_{u \in \mathcal{P}} \sum_{i=1}^{len(u)} \mathbb{1}_{gr(l_u^i) = gr(\hat{l}_u^i)}}{\sum_{u \in \mathcal{P}} len(u)} \tag{4.11}$$

Normalized Distance Error (NDE). Let $dist(l_u^i, \hat{l}_u^i)$ denote the Euclidean distance between locations l_u^i and $\hat{l}_u^i$, and let d_{max} denote the maximum possible distance that exists in the corresponding dataset:

$$d_{max} = \sqrt{(max\_lat - min\_lat)^2 + (max\_long - min\_long)^2} \tag{4.12}$$

where max_lat and min_lat are the maximum and minimum latitudes, and max_long and min_long are the maximum and minimum longitudes in the dataset. Then, Normalized Distance Error (NDE) measures the average distance between l_u^i and $\hat{l}_u^i$ across all users and predictions, normalized according to d_{max}:

$$\text{Normalized Distance Error (NDE)} = \frac{\sum_{u \in \mathcal{P}} \frac{\sum_{i=1}^{len(u)} dist(l_u^i, \hat{l}_u^i)}{d_{max} \cdot len(u)}}{|\mathcal{P}|} \tag{4.13}$$

The rationale behind normalization using d_{max} is because different datasets cover geographic regions with different sizes (e.g., max_lat, min_lat, max_long and min_long are quite different). Normalization is performed so that results are consistent and comparable between datasets.

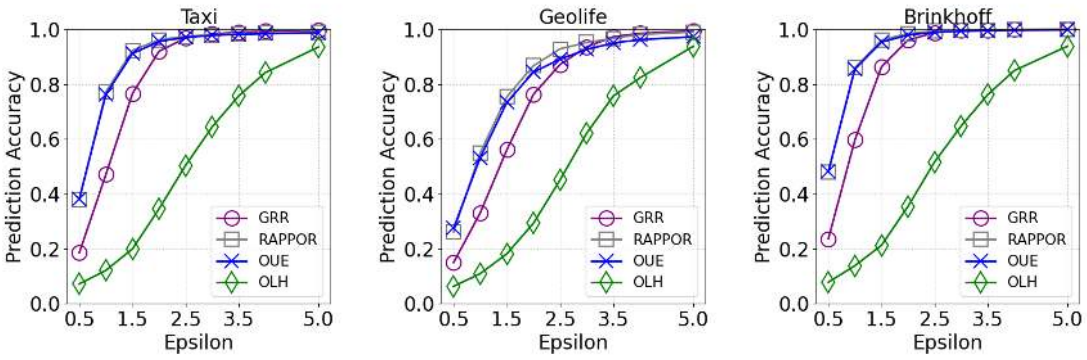

Figure 4.2: Prediction accuracies of our statistical attacks against stationary users

4.4.2 Results of Our Attacks Against Stationary Users

Originally, users in our three datasets (Taxi, Geolife, Brinkhoff) are mobile. To test our statistical attacks against stationary users, we simulated stationary users for each of the three datasets by taking the first location l_u^1 of each user and repeating it $len(u)$ many times. Then, we executed our statistical attacks for different LDP protocols (GRR, RAPPOR, OUE, OLH) and different ε values ranging from 0.5 to 5. The results are shown in Figure 4.2.

We observe from Figure 4.2 that our attacks achieve remarkably high prediction accuracies under different protocols and realistic ε budgets, such as $\varepsilon = 1$ and 2. On Taxi and Brinkhoff datasets, PA is close to 80% for RAPPOR and OUE even with $\varepsilon = 1$, meaning that 80% of the attacker's predictions will be correct. Furthermore, for $\varepsilon \geq 3$, PA reaches close to 100% under RAPPOR, OUE and GRR, showing that the users' locations are almost always correctly predicted by the attacker. These results demonstrate the effectiveness of our statistical attacks against stationary users. We also observe that in general, PA of RAPPOR and OUE are highest, followed by GRR, and finally by OLH. The reason behind OLH's lower PA is that OLH's hashing provides an additional privacy benefit. Even if the user's reported integer between $[1, k]$ can be correctly recovered by the attacker, there can be hash collisions, i.e., multiple cells that hash to the same integer. Since the attacker cannot resolve this hash collision, s/he must perform a random guess among the colliding

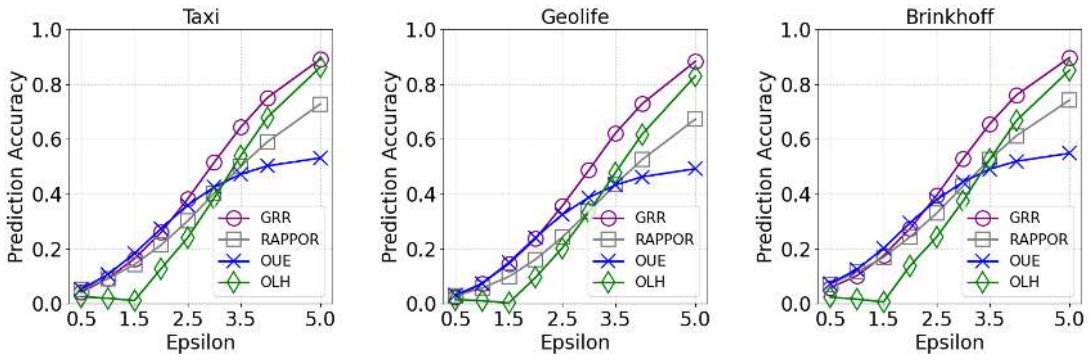

Figure 4.3: Prediction accuracies of our HMM-based attacks against mobile users

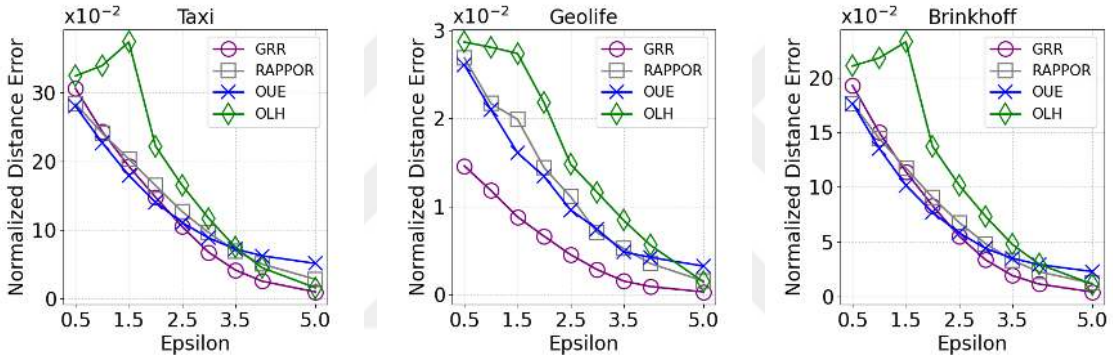

Figure 4.4: NDEs of our HMM-based attacks against mobile users

values, which results in lower PA.

4.4.3 Results of Our HMM-Based Attacks

The results of our HMM-based attacks on the three datasets are shown in Figures 4.3 and 4.4 for the PA and NDE metrics, respectively. Note that these attacks do not contain the extensions (informed and chain). As expected, when users are mobile rather than stationary, it is more difficult to correctly predict their true locations. Hence, PAs in Figure 4.3 are generally lower than Figure 4.2. However, our attacks are still quite effective for high values of ε, such as $\varepsilon \geq 3$. In general, the amount of increase in PA is close to linear in terms of increasing ε.

Overall, the GRR protocol seems to be most vulnerable against the HMM-based

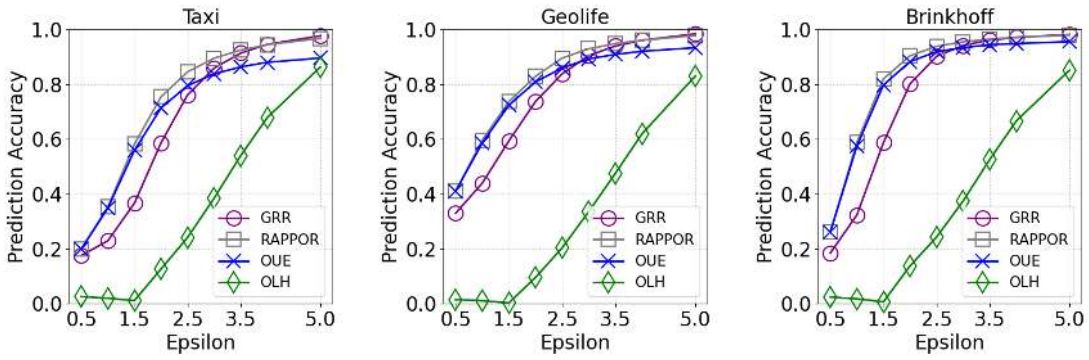

Figure 4.5: Prediction accuracies of our informed attacks against mobile users

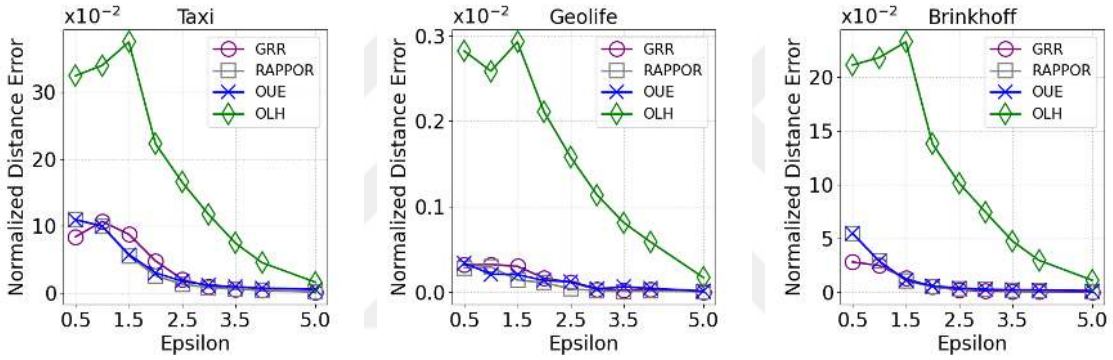

Figure 4.6: NDEs of our informed attacks against mobile users

attack, in terms of both PA and NDE metrics. OLH protocol exhibits lower PA and higher NDE for low ε values, but as ε increases, its PA exceeds RAPPOR and OUE. The reason behind this is because of the k value used in OLH. Since k depends on ε, as we increase ε, the value of k also increases, yielding fewer hash collisions. With fewer hash collisions, it is easier for our attack to achieve higher PA and lower NDE. The two bitvector-based protocols, RAPPOR and OUE, are also usually more robust than GRR. The reason behind this is because in OUE and RAPPOR, the number of observable states is much higher compared to other protocols. Thus, the emission probabilities are lower. Combining these two factors, the Viterbi algorithm yields less accurate inferences.

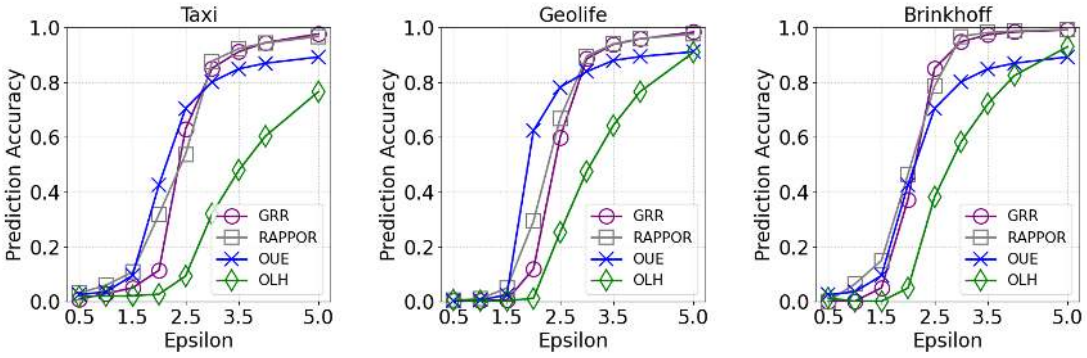

Figure 4.7: Prediction accuracies of our chain attacks against mobile users

4.4.4 Results of Attack Extensions

Results of informed attacks. We illustrate the results of our informed attacks in Figures 4.5 and 4.6 using the PA and NDE metrics. Comparing Figures 4.5 and 4.6 with Figures 4.3 and 4.4, we observe that the informed attacks perform significantly better than the plain HMM-based attacks. For $\varepsilon \leq 2.5$ and for GRR, RAPPOR, and OUE protocols, the PAs of informed attacks are almost twice as high as the PAs of the earlier attacks. Furthermore, NDEs are also much lower (one third or less). For $\varepsilon > 2.5$, PAs of the informed attack become close to 100%, which means the attacker is able to predict almost all locations of the users successfully. The OLH protocol lags behind GRR, RAPPOR and OUE in terms of PA and NDE, again due to its hash-based nature. Nevertheless, PA and NDE of OLH are still higher for the informed attack.

The reason behind the informed attacks outperforming the plain HMM-based attacks is because in the plain HMM-based attacks, Π and $\mathcal{A}$ are populated using uniform distributions. In contrast, aggregate population-level statistics are used to populate Π and $\mathcal{A}$ in the informed attacks. Thus, we can conclude that a real-world attacker can significantly increase his/her attack effectiveness by incorporating aggregate population-level statistics.

Results of chain attacks. The results of our chain attacks are shown in Figures 4.7 and 4.8. In these figures, we used $\alpha = 3$ as the number of added iterations.

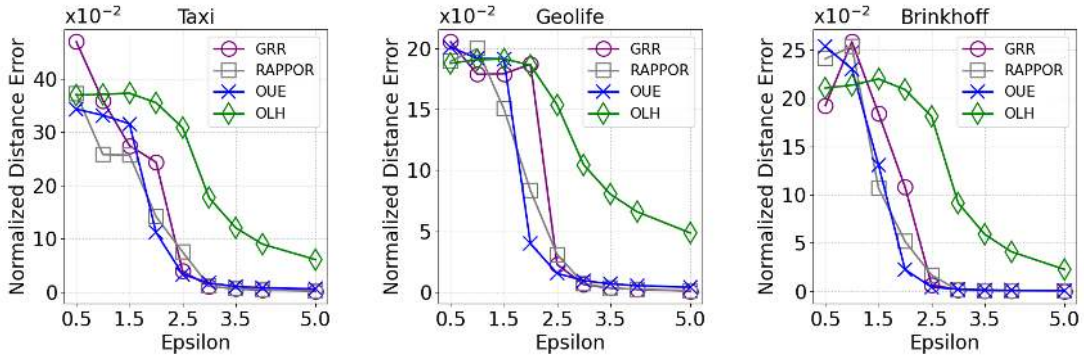

Figure 4.8: NDEs of our chain attacks against mobile users

Notably, for low ε values such as $\varepsilon \leq 1.5$, the PA of chain attacks are lower than the plain HMM-based attacks, and the NDEs of chain attacks are higher than the plain HMM-based attacks. Hence, we can conclude that for low ε, chain attacks are less effective than our plain HMM-based attacks. In contrast, when ε is high (such as $\varepsilon \geq 3.5$), chain attacks yield higher PA and lower NDE compared to plain attacks; therefore they become more effective. This phenomenon regarding the chain attack is a result of the model's attempt to learn from its own predictions. As noted in Section 4.3, when previous predictions are accurate, such learning makes the attack even more accurate. Yet, when previous predictions contain errors, learning causes errors to propagate and amplify. We observe from Figures 4.7 and 4.8 that this is indeed the case – when ε is low, predictions are less accurate therefore the chain attack suffers. When ε is high, predictions are more accurate therefore the chain attack performs better.

In Tables 4.2 and 4.3, we experiment with varying values of α used in the chain attack to observe how α impacts attack effectiveness. We perform this experiment on the Taxi dataset but results on the remaining datasets are similar. $\varepsilon = 3$ and 3.5 are used as representative values, since these ε values are towards the middle of the ε range we used in our previous experiments; in addition, it is sensible to perform this experiment in the region where the chain attack outperforms the plain HMM-based attack (i.e., not low ε). We observe from the results in Tables 4.2 and 4.3 that PAs can vary significantly depending on the value of α. When α is too low

Table 4.2: Prediction accuracies of the chain attack under varying α (Taxi dataset, $\varepsilon = 3$).

α	GRR	RAPPOR	OUE	OLH
1	**0.628**	0.807	0.814	0.798
3	0.537	**0.850**	**0.824**	**0.817**
5	0.571	0.732	0.754	0.766
7	0.409	0.438	0.483	0.477

Table 4.3: Prediction accuracies of the chain attack under varying α (Taxi dataset, $\varepsilon = 3.5$).

α	GRR	RAPPOR	OUE	OLH
1	**0.807**	0.891	0.879	0.839
3	0.728	**0.895**	**0.891**	**0.859**
5	0.631	0.803	0.822	0.812
7	0.562	0.583	0.582	0.615

or too high, PAs can be low. $\alpha = 3$ indeed yields the highest PA for RAPPOR, OUE and OLH protocols. In contrast, highest PA for the GRR protocol is obtained with $\alpha = 1$. The reason why we used $\alpha = 3$ in Figures 4.7 and 4.8 is because of the overall high performance of $\alpha = 3$ compared to other α, as demonstrated in Tables 4.2 and 4.3. We note that for higher ε (such as close to 5), higher α (e.g., $\alpha = 5$) can yield better results.

4.4.5 Impact of Trajectory Lengths

Next, we evaluate the impact of trajectory lengths (i.e., the number of observations $n = |\mathcal{O}_u|$) on the effectiveness of our attacks. For each different trajectory length n, we find those users in the datasets that have trajectories with lengths of at least n. For users with trajectories longer than n, we keep the first n locations and discard the rest. We repeat this experiment on multiple datasets using the informed versions

of our attacks. We report the results in Figure 4.9 for the statistical attacks and in Figure 4.10 for the HMM-based attacks. In both figures, we fix the ε value to $\varepsilon = 2.5$.

First, we observe that in all cases, increasing trajectory lengths cause the attacks to become more effective. The main reason is that increasing lengths enable the attacker to obtain more observations from the same user. Although each observation is protected by LDP, the correlations between the observations enable more effective attacks. Second, we observe that the PA increases in statistical attacks are more pronounced than the PA increases in HMM-based attacks. This is because LDP outputs from stationary users exhibit the highest correlation between consecutive timestamps. In contrast, although correlations between consecutive timestamps exist in the case of mobile users, the user is likely to be moving, therefore it is less likely that the identical location cell will be reported via LDP. An experiment result that strengthens this explanation is the fact that in Figure 4.10, PAs on the Geolife dataset are higher than Brinkhoff. Upon comparing users' characteristics in Brinkhoff and Geolife, we observed that users in Brinkhoff are mobile (moving more frequently and traveling larger distances) compared to Geolife. Finally, we note that the results in Figures 4.9 and 4.10 agree with the previous results in terms of how the different LDP protocols compare with one another. PAs are similar for GRR, RAPPOR and OUE protocols, whereas PAs for the OLH protocol are generally lower.

4.4.6 Executing Stationary Attacks on Mobile Users and HMM-Based Attacks on Stationary Users

Recall that we originally designed statistical attacks against stationary users and HMM-based attacks against mobile users. In this section, in order to analyze how the attacks perform in scenarios in which they were *not* originally designed for, we execute statistical attacks on mobile users and HMM-based attacks on stationary users.

The results are reported in Table 4.4. Upon comparing the statistical and HMM-

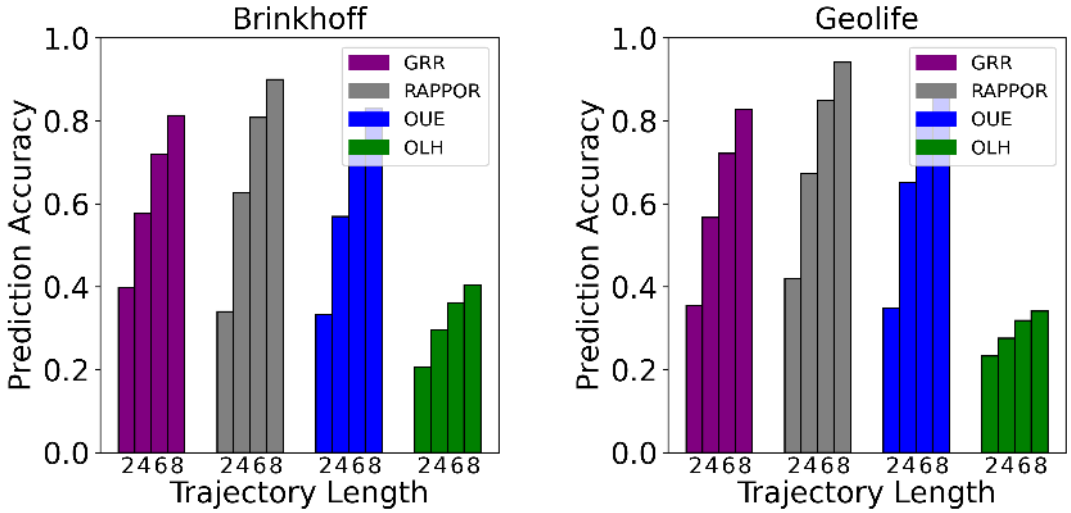

Figure 4.9: Prediction accuracies of our statistical attacks against stationary users, under varying trajectory lengths (fixed $\varepsilon = 2.5$).

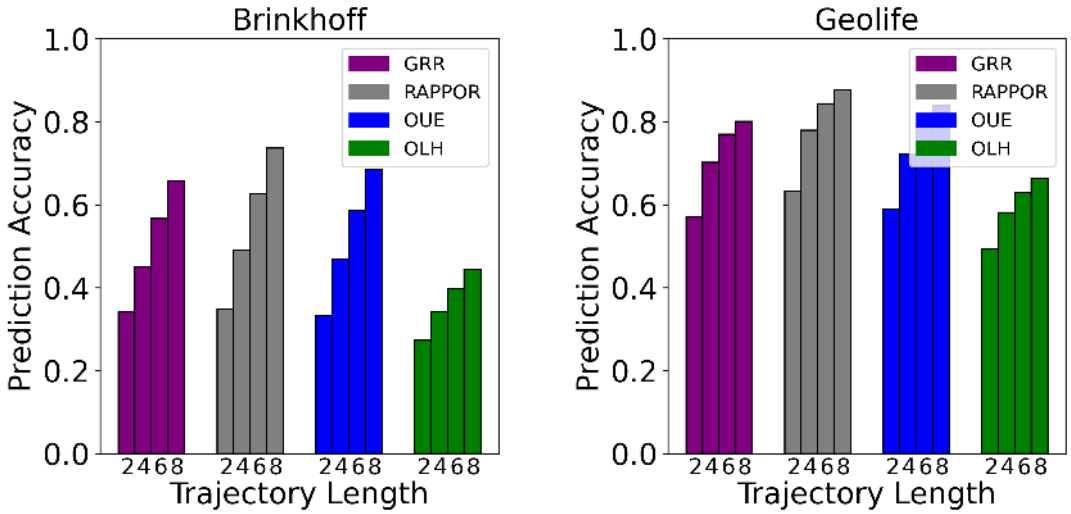

Figure 4.10: Prediction accuracies of our HMM-based attacks against mobile users, under varying trajectory lengths (fixed $\varepsilon = 2.5$).

based attacks on stationary users, we observe that the stationary attack always outperforms the HMM-based attack by a large margin. PAs of the statistical attack quickly reach ≥ 0.9 when $\varepsilon \geq 2$, while PAs of the HMM-based attack typically

Table 4.4: Prediction accuracies of executing the statistical attack and HMM attack on both stationary users and mobile users (results are for the Taxi dataset).

		Stationary Users		Mobile Users	
Protocol	ε	Stat. Att.	HMM Att.	Stat. Att.	HMM Att.
GRR	1	0.471	0.096	0.153	0.092
	2	0.919	0.264	0.333	0.259
	3	0.983	0.517	0.410	0.515
	4	0.993	0.751	0.426	0.750
	5	0.998	0.892	0.433	0.891
RAPPOR	1	0.772	0.100	0.222	0.091
	2	0.962	0.229	0.358	0.214
	3	0.980	0.418	0.405	0.401
	4	0.986	0.604	0.420	0.589
	5	0.990	0.735	0.427	0.728
OUE	1	0.762	0.116	0.224	0.106
	2	0.956	0.285	0.361	0.273
	3	0.978	0.430	0.404	0.425
	4	0.984	0.500	0.419	0.502
	5	0.987	0.526	0.425	0.531
OLH	1	0.121	0.018	0.059	0.019
	2	0.345	0.125	0.123	0.127
	3	0.644	0.357	0.278	0.384
	4	0.842	0.640	0.364	0.679
	5	0.936	0.837	0.408	0.863

remain much lower (between 0.1 and 0.8). On the other hand, upon comparing the statistical and HMM-based attacks on mobile users, we observe that in the majority of cases, the HMM-based attack outperforms the stationary attack. This is true especially for larger ε values, such as $\varepsilon \geq 3$. For smaller ε values, it is possible that the stationary attack has similar or slightly higher PA than the HMM-based attack. But overall, our take-away message is that the attacks are more suited for the scenario that they were designed for, i.e., statistical attacks (designed for stationary scenarios) achieve higher PA than HMM-based attacks in case of stationary users, whereas HMM-based attacks (designed for mobile scenarios) achieve higher PA than statistical attacks in case of mobile users.

Chapter 5

EFFICIENCY IMPROVEMENTS FOR ATTACKS ON BITVECTOR-BASED PROTOCOLS

In the previous chapter, we demonstrated that our HMM-based attacks are effective on multiple LDP protocols, including bitvector-based protocols such as RAPPOR and OUE. On the other hand, we observed that our HMM-based attacks suffer from two inefficiency problems when applied to bitvector-based protocols: high memory usage and high execution times. These hinder the scalability of our attacks especially when the grid size $|\mathcal{G}|$ is large. In this chapter, we aim to address these inefficiency problems by proposing heuristic improvements.

5.1 Main Causes of Inefficiency

First, we identify the main causes behind high memory usage and execution times. Recall that our HMM-based attacks contain five components: O, S, $\mathcal{A}$, $\mathcal{B}$, and Π. Among them:

- The set of hidden states S correspond to the cells in $\mathcal{G}$. It is not dependent on the LDP protocol, i.e., the contents of S are the same for all LDP protocols. Since each $C_i \in \mathcal{G}$ is represented by one hidden state in S, the size of S grows linearly with the growth of $\mathcal{G}$.

- The initial probability distribution Π is also protocol-independent and its length also grows linearly with the growth of $\mathcal{G}$.

- The transition probability matrix $\mathcal{A}$ contains the probability of moving from each cell to another, e.g., $\mathcal{A}[i, j]$ denotes the probability that the user will move from cell C_i to C_j. Although $\mathcal{A}$ is protocol-independent, it grows quadratically with the growth of $\mathcal{G}$, since the size of the matrix is $|\mathcal{G}| \times |\mathcal{G}|$.

The two remaining components of HMMs are the set of observable states O and the emission probability matrix $\mathcal{B}$. These components are protocol-dependent. For bitvector-based LDP protocols such as RAPPOR and OUE:

- The set of observable states O corresponds to the set of observable bitvectors. Since the length of the users' bitvectors is equal to $|\mathcal{G}|$, the size of O is potentially $2^{|\mathcal{G}|}$. This implies exponential growth. Furthermore, for typical values of $|\mathcal{G}|$ such as 20 and 25, the size of O becomes quite large.

- The emission probability matrix $\mathcal{B}$ contains a probability from each hidden state S to each observable state O. Thus, the size of $\mathcal{B}$ is $|O| \times |S| = 2^{|\mathcal{G}|} \times |\mathcal{G}|$. This again implies exponential growth with respect to $|\mathcal{G}|$.

As we can see from the discussion above, the protocol-dependent components of our HMM-based attacks (O and $\mathcal{B}$) both grow exponentially in the case of bitvector-based protocols RAPPOR and OUE. These not only require larger memory in the Python code, but also increase the attack execution time due to the increased construction time of the components, as well as the increased time to execute the Viterbi algorithm. Such large growths are not observed for other protocols like GRR and OLH. For example, in GRR we have $|O| = |\mathcal{G}|$, and in OLH we have $|O| = k$ because of the protocols' definitions. Hence, the inefficiency problems only occur in bitvector-based protocols RAPPOR and OUE. As $|\mathcal{G}|$ grows, certain attack executions that take a few minutes on GRR and OLH protocols can take multiple hours on RAPPOR and OUE protocols.

5.2 Proposed Efficiency Improvements

We propose heuristic methods to solve the aforementioned inefficiency problems. The key idea behind our solution is to remove those states from O that have very low probabilities of being observed in practice while keeping those states in O that have larger observation probabilities. Consequently, $|O|$ will be reduced, which causes $|\mathcal{B}|$ to also reduce.

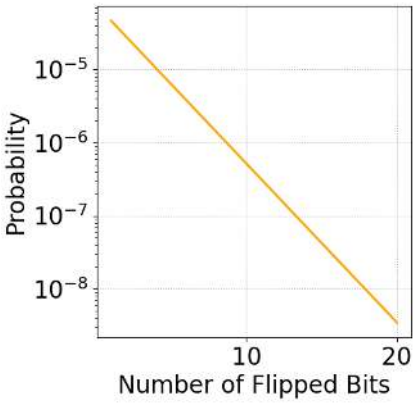

Figure 5.1: Probability of occurrence for an observed state versus the number of flipped bits in that state (RAPPOR protocol, $\varepsilon = 1$, $|\mathcal{G}| = 20$).

To demonstrate the intuition behind our solution, we perform the following experiment. We construct a hypothetical O using the RAPPOR protocol, $\varepsilon = 1$ and $|\mathcal{G}| = 20$. Recall from Section 3.3 that each observable state $b_j \in O$ denotes a bitvector. For each b_j, we count the number of flipped bits (1 bits) in b_j:

$$\text{Number of Flipped Bits} = \sum_{t=1}^{|b_j|} b_j[t] \tag{5.1}$$

Then, we measure the average probability of occurrence for b_j:

$$\text{Probability of Occurrence} = \frac{\sum_{i=1}^{|O|} \mathcal{B}[i,j]}{|O|} \tag{5.2}$$

In Figure 5.1, we plot the probability of occurrence versus the number of flipped bits for different states b_j. One can observe that as the number of flipped bits increases, the probability of observing that state decreases dramatically, e.g., multiple orders of magnitude. In other words, states in O which have few 1 bits (e.g., between 0 and 5 one bits) are multiple orders of magnitude more likely compared to states which have many 1 bits (e.g., between 15-20 one bits). This is an intuitive result, considering the definition of the RAPPOR protocol. Users' original bitvectors (before perturbation) contain one 1 bit, and the probability of keeping the original

bits is $\frac{e^{\varepsilon/2}}{e^{\varepsilon/2}+1}$ whereas the probability of changing a 0 bit to a 1 bit is $\frac{1}{e^{\varepsilon/2}+1}$. Thus, observing a state in O that contains many 1 bits is far less likely than observing a state that contains few 1 bits.

Considering the aforementioned intuition and result, our solution to improve attack efficiency is to remove those states in O which have low probabilities of occurrence, i.e., those states in O which contain many 1 bits. More specifically, for each state $b_j \in O$, if the Number of Flipped Bits in b_j is greater than θ, we remove b_j from O. However, we note that removing states from O causes problems in the emission matrix $\mathcal{B}$. For a hidden state s_i, the sum of probabilities of emitting all observed states should be 1, i.e.: $\sum_j \mathcal{B}[i,j] = 1$. This is true in our original HMM-based attack, but when we remove states b_j from O, it no longer holds. To fix this problem, we go through the emission matrix $\mathcal{B}$ row by row and ensure each row adds up to 1. For each row i, we find its sum after states are removed:

$$sum\_prob = \sum_j \mathcal{B}[i,j] \tag{5.3}$$

Then, we update each value in that row $\mathcal{B}[i,j]$ by $\mathcal{B}[i,j] \times \frac{1}{sum\_prob}$. This ensures that the sum of that row will add up to 1.

The above multiplication ensures that row-wise summations in $\mathcal{B}$ add up to 1, and therefore the emission matrix $\mathcal{B}$ becomes mathematically legitimate. However, it causes a reduction in the accuracy of $\mathcal{B}$ since the probabilities of existing states are increased heuristically to make up for the lost probabilities of the removed states. As a result, the accuracy of our attacks will decrease, both because of the removal of states, and also because of heuristically increased probabilities of remaining states. This causes a trade-off between attack efficiency and effectiveness, i.e., while our methods improve attack efficiency, they are likely to have negative impacts on attack effectiveness. Intuitively, higher the number of removed states, higher the improvement in efficiency, but also higher the impact on attack effectiveness.

Table 5.1: Impact of varying θ on attack PA, memory usage, and execution time (OUE protocol, $\varepsilon = 2$).

Value of θ	**PA**	**Memory (MB)**	**Time (minutes)**
$\theta = \|\mathcal{G}\|$	0.27	160.6	303.5
$\theta = \|\mathcal{G}\|/2$	0.27	93.0	17.0
$\theta = \|\mathcal{G}\|/4$	0.26	8.3	9.0
$\theta = \|\mathcal{G}\|/8$	0.14	1.1	7.7
$\theta = \|\mathcal{G}\|/16$	0.09	0.6	7.5

5.3 Choosing the Value of θ

Considering that θ determines how many states will be removed, it has a direct impact on attack efficiency and effectiveness. Therefore, it is important to choose a good value for the θ parameter. In order to find a good value for the θ parameter, we conduct a series of experiments with different θ under varying ε and the two bitvector-based protocols (RAPPOR and OUE). In each experiment, we note the Prediction Accuracy (PA) of our attack, as well as its memory requirement (how much memory it uses in the computer's RAM) and execution time. A subset of our experiment results are provided in Tables 5.1 and 5.2; remaining results are not provided for brevity, but they contain similar take-away messages. To conduct this experiment, we utilized Python's tracemalloc library to monitor memory usage of the algorithms and the time library to measure execution time.

The two tables show similar trends. As we decrease θ from $|\mathcal{G}|$ to $|\mathcal{G}|/2$ and $|\mathcal{G}|/4$, PAs of our attacks decrease slightly, e.g., by 0 - 0.01 in Table 5.1 and by 0.03 - 0.05 in Table 5.2. On the other hand, we achieve substantial efficiency improvements in terms of memory usage and execution time. When $\theta = |\mathcal{G}|/4$, the memory usage is lower by approximately 20 folds and execution time is lower by approximately 35 folds compared to $\theta = |\mathcal{G}|$. Hence, we conclude that using $\theta = |\mathcal{G}|/2$ or $\theta = |\mathcal{G}|/4$ are good choices for the θ parameter, since they cause limited reductions in attack effectiveness (i.e., limited reductions in PA) while achieving substantial reductions

Table 5.2: Impact of varying θ on attack PA, memory usage, and execution time (RAPPOR protocol, $\varepsilon = 3$).

Value of θ	**PA**	**Memory (MB)**	**Time (minutes)**
$\theta = \|\mathcal{G}\|$	0.40	160.6	303.5
$\theta = \|\mathcal{G}\|/2$	0.37	93.0	16.6
$\theta = \|\mathcal{G}\|/4$	0.35	8.3	8.3
$\theta = \|\mathcal{G}\|/8$	0.07	1.1	7.3
$\theta = \|\mathcal{G}\|/16$	0.05	0.6	6.7

in memory usage and execution time.

Next, we consider even lower values of θ, such as $\theta = |\mathcal{G}|/8$ and $\theta = |\mathcal{G}|/16$. We observe from Tables 5.1 and 5.2 that memory usage can be reduced to around 1 MB and the execution time can become less than 7-8 minutes when these θ values are used. Yet, these θ values cause substantial reductions in PA, i.e., PAs can reduce by two to five folds. Such reductions in attack effectiveness are high prices to pay for improved memory usage and execution time. Thus, we do not recommend low values of θ such as $\theta = |\mathcal{G}|/8$ and $\theta = |\mathcal{G}|/16$.

Overall, based on our experiment results, we recommend the use of $\theta = |\mathcal{G}|/4$ by default, since: (i) it offers a good trade-off between attack effectiveness and efficiency by having a limited impact on PA but substantial reduction in memory usage and execution time, and (ii) its memory usage around 8-9 MBs and execution time less than 10 minutes imply that the attacks become practical and scalable in practice.

5.4 Experiment Results

In this section, we experimentally compare our new attacks (with efficiency improvement, $\theta = |\mathcal{G}|/4$) versus our old attacks from the previous chapter (without efficiency improvement). We use the same experiment setup from Section 4.4. All experiments were conducted on a Windows 11 computer with an Intel i7 10700K CPU and 32 GB RAM. First, we compare the execution times and memory usage

Table 5.3: Average execution times of our old HMM-based attacks from the previous chapter (without efficiency improvement) and new attacks (with efficiency improvement, $\theta = |\mathcal{G}|/4$). Results are reported in minutes.

Dataset	**RAPPOR (Old)**	**RAPPOR (New)**	**OUE (Old)**	**OUE (New)**
Taxi	306.8	8.1	303.5	9.1
Geolife	38.1	1.1	44.1	1.2
Brinkhoff	51.4	2.3	57.6	2.5

Table 5.4: Average memory usages of our old HMM-based attacks from the previous chapter (without efficiency improvement) and new attacks (with efficiency improvement, $\theta = |\mathcal{G}|/4$). Results are reported in megabytes.

Dataset	**RAPPOR (Old)**	**RAPPOR (New)**	**OUE (Old)**	**OUE (New)**
Taxi	160.61	8.31	160.61	8.32
Geolife	160.57	8.28	160.57	8.28
Brinkhoff	160.56	8.27	160.56	8.27

of our old versus new attacks under varying datasets (Taxi, Geolife, Brinkhoff) and ε values. After performing experiments with various ε, we took their average since varying the ε value has a negligible impact on execution times and memory usage. The results are reported in Tables 5.3 and 5.4.

Tables 5.3 and 5.4 demonstrate that our new methods substantially reduce both the average execution times and memory usage. Since RAPPOR and OUE are both bitvector-based protocols, their results are similar. Without the methods in this chapter, our attacks took close to 5 hours on the Taxi dataset, 0.5 - 0.75 hours on the Geolife dataset, and close to an hour on the Brinkhoff dataset. With the methods explained in this chapter, we can now conduct the attacks within 8-9 minutes on the Taxi dataset, and within 1-3 minutes on the Geolife and Brinkhoff datasets. While attacks that took an hour or longer can be prohibitive for an attacker in the real world, attacks that take a few minutes are much more practical. Hence, we

Table 5.5: Execution times of our old versus new attacks on the Taxi dataset with varying $|\mathcal{G}|$.

Value of $\|\mathcal{G}\|$	**RAPPOR (Old)**	**RAPPOR (New)**	**OUE (Old)**	**OUE (New)**
$\|\mathcal{G}\| = 5$	2.0 mins	2.5 mins	2.3 mins	2.8 mins
$\|\mathcal{G}\| = 10$	3.7 mins	3.9 mins	3.5 mins	4.4 mins
$\|\mathcal{G}\| = 15$	11.8 mins	5.5 mins	9.3 mins	6.1 mins
$\|\mathcal{G}\| = 20$	5.1 hours	8.4 mins	5.1 hours	9.1 mins
$\|\mathcal{G}\| = 25$	$>$ 1 day	64.2 mins	$>$ 1 day	39.3 mins
$\|\mathcal{G}\| = 30$	$>$ 1 day	7.9 hours	$>$ 1 day	8.5 hours

achieve substantial improvement in the practicality of our attacks. We note that the difference between the datasets stems from the dataset size (number of users per dataset) as well as the average trajectory length in the dataset. As explained in Section 4.4, Taxi is a much larger dataset than Geolife and Brinkhoff, which is why attacks on Taxi take longer in general.

In terms of memory usage, Table 5.4 shows that old attacks on RAPPOR and OUE typically use close to 160 MBs of memory, whereas new attacks use close to 8 MBs of memory. This translates to roughly 20-fold reduction in memory usage.

Next, we compare the scalability of our new attacks versus old attacks under varying $|\mathcal{G}|$. We perform this experiment on the Taxi dataset since it is our largest dataset. We vary $|\mathcal{G}|$ between 5 and 30 in increments of 5, and measure the execution times and memory usage of the attacks. Larger $\mathcal{G}$ implies that the grid will contain a higher number of cells, thereby increasing the precision of users' locations and the computational resources required by the attacker, since many components of the HMM-based attacks grow linearly or super-linearly with the growth of $\mathcal{G}$.

The results are reported in Tables 5.5 and 5.6. We observe from Table 5.5 that the old attacks are feasible when $|\mathcal{G}| \leq 15$, considering that they finish within a few minutes. However, their execution times become in the order of hours as $|\mathcal{G}|$ becomes 20, and furthermore, their executions do not finish within a day when $|\mathcal{G}| \geq 25$. In contrast, the new attacks always finish within several minutes or hours for all $|\mathcal{G}|$

Table 5.6: Memory usage of our old versus new attacks on the Taxi dataset with varying $|\mathcal{G}|$. Results are reported in megabytes. (DNF = Did Not Finish in 1 day)

Value of $\|\mathcal{G}\|$	**RAPPOR (Old)**	**RAPPOR (New)**	**OUE (Old)**	**OUE (New)**
$\|\mathcal{G}\| = 5$	0.61	0.61	0.60	0.61
$\|\mathcal{G}\| = 10$	0.68	0.64	0.68	0.64
$\|\mathcal{G}\| = 15$	4.36	0.93	4.36	0.93
$\|\mathcal{G}\| = 20$	160.61	8.31	160.61	8.32
$\|\mathcal{G}\| = 25$	DNF	106.85	DNF	106.85
$\|\mathcal{G}\| = 30$	DNF	1169.29	DNF	1169.29

between 5 and 30, demonstrating the scalability benefits of the methods proposed in this chapter. In terms of memory usage, we observe from Table 5.6 that there is no significant difference between the memory usage of the old versus new attacks for small $|\mathcal{G}|$ such as $|\mathcal{G}| \leq 10$. On the other hand, as $|\mathcal{G}|$ becomes 15 or higher, the differences between the old and new attacks become more significant, with the new attacks having substantially lower memory usage. Overall, despite significant improvements achieved by our new attacks, we should still note that $|\mathcal{G}| \geq 35$ will likely be computationally demanding, both in terms of execution time and memory usage, based on our current results and findings. Thus, further scalability improvements in addition to those presented in this chapter may be desirable for $|\mathcal{G}| \geq 35$. We leave this as an open area for future work.

Finally, we explore the PAs of the old versus new attacks. From an attacker's perspective, it is desirable that the new attacks maintain as high PA as possible compared to the old attacks, since this would mean that the efficiency improvements do not hurt attack effectiveness. In Figure 5.2, we report the PAs of the old versus new attacks under varying datasets and ε values. In general, we observe from Figure 5.2 that the PAs of the new attacks are quite similar to the PAs of the old attacks. The results of the old and new attacks are especially similar (almost identical) for the OUE protocol. For the RAPPOR protocol, there seem to be small differences between the old and new attacks when $\varepsilon \leq 3$; however, these differences are reduced

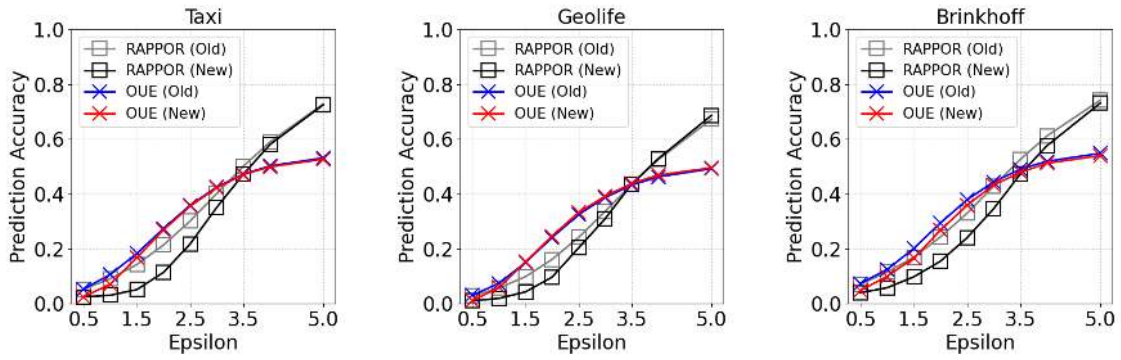

Figure 5.2: Prediction accuracies (PA) of our old versus new attacks on three datasets and varying ε values.

and the PAs become almost identical when $\varepsilon \geq 3.5$.

Chapter 6

PROPOSED DEFENSES

In this chapter, we propose three defense strategies against our attacks: Memoization, Replay, and Replication. Then, we experimentally evaluate the three defenses and discuss their effectiveness, utility, and efficiency implications.

6.1 Memoization

The concept of memoization in LDP was originally introduced in RAPPOR [Erlingsson et al., 2014] to defend against longitudinal attacks. It was later adapted in works such as [Arcolezi et al., 2022] to improve LDP protocols for longitudinal and multidimensional frequency estimation. In our work, we adapt and apply the memoization idea to multiple LDP protocols in our context of continuous location sharing.

Before explaining our adaptation of memoization to our context, we explain the intuition behind memoization. Let v^t denote the user's true value at time t. Without memoization, the user would perturb v^t with an LDP protocol and send the output $\Psi(v^t)$ to the data collector. Instead, with memoization, the user first computes $\Psi(v^t)$ and locally stores (memoizes) it. Let $mem \leftarrow \Psi(v^t)$ denote this memoized value. In the current timestamp t and future consecutive timestamps $t+1$, $t+2$, ... in which the user's true value does not change (i.e., remains equal to v^t), the user freshly perturbs mem with LDP and sends the output $\Psi(mem)$ to the data collector. The rationale behind this second perturbation is that an attacker who observes many perturbed $\Psi(mem)$ values will be able to run a longitudinal attack (similar to our attacks), but instead of inferring v^t, the attacker will only be able to infer mem because the attacker's observations are perturbed versions of mem rather than perturbed versions of v^t. Since $mem = \Psi(v^t)$, even if the attacker correctly

Algorithm 2 Memoization Defense

Input: User u's real locations $l_u^1, l_u^2, ..., l_u^n$

Output: User u's reported values $\tilde{l}_u^1, \tilde{l}_u^2, ..., \tilde{l}_u^n$

$mem \leftarrow \Psi(l_u^1)$
$\tilde{l}_u^1 \leftarrow \Psi(mem)$
for $t = 2$ to n **do**
  if $l_u^t = l_u^{t-1}$ **then**
    $\tilde{l}_u^t \leftarrow \Psi(mem)$
  else
    $mem \leftarrow \Psi(l_u^t)$
    $\tilde{l}_u^t \leftarrow \Psi(mem)$
  end if
end for
return $\tilde{l}_u^1, \tilde{l}_u^2, ..., \tilde{l}_u^n$

infers mem, v^t remains protected by the privacy achieved by Ψ.

Our adaptation of memoization to continuous location sharing is shown in Algorithm 2. Recall that for user $u \in \mathcal{P}$, the user's true location at time t is denoted by l_u^t. For the first location l_u^1, since there can be no location prior to it, the user applies the first perturbation $\Psi(l_u^1)$, stores the result as mem, and reports the result of the second perturbation $\Psi(mem)$ to the data collector. For remaining locations $t = 2$ onwards, the algorithm checks if the current location l_u^t is equal to the previous location l_u^{t-1}. If so, the previously stored mem is retrieved and perturbed using LDP, i.e., $\Psi(mem)$ is obtained, and this result is sent to the data collector. However, if $l_u^t \neq l_u^{t-1}$, then previous mem is discarded and the current mem is freshly computed with the current location (i.e., $mem \leftarrow \Psi(l_u^t)$). Then, the fresh mem is perturbed using Ψ to obtain the result of the second perturbation, and this result is sent to the data collector.

The memoization defense has advantages and disadvantages. As the results will show, its main advantage is the substantial and consistent reduction in PA, which demonstrates that it is an effective defense. The effectiveness of memoization is

intuitive: (i) although the longitudinal attack may correctly infer mem, l_u^t remains protected by the LDP achieved by Ψ, and (ii) even if longitudinal attacks were not an issue, e.g., the user reported only one location or consecutive locations were completely uncorrelated, since the user's reported value is the result of two LDP perturbations, the attacker's PA is naturally decreased. On the other hand, the main disadvantage of memoization is the utility loss caused by performing two perturbations instead of one. Another disadvantage of memoization is having to store mem securely on the user's device, which brings an additional storage space consumption and a secure storage requirement.

6.2 Replay

Our replay defense aims to defend against longitudinal attacks using a *single* step of perturbation as opposed to two perturbations performed by memoization, thereby circumventing memoization's utility loss problem. In replay, the user maintains a private key-value store called the *cache* on his/her device. This cache stores the user's previous locations and the corresponding perturbed counterparts (with LDP) of each location. When a certain location is repeated, the user retrieves the perturbed counterpart of this location from the cache and replays it to the data collector. If the location is new, i.e., it is not found in the cache, then it will be freshly perturbed and sent to the data collector, and the value that was sent to the data collector will be written to the cache.

The pseudocode of our replay defense is shown in Algorithm 3. The *cache* stores key-value pairs of the form $\langle l_u, val \rangle$ where l_u is a location and val is its corresponding counterpart perturbed with LDP. The *cache* is initialized as empty. For each location l_u^t from $t = 1$ to n, the user first checks if a key-value pair with key equal to l_u^t exists in the cache. If so, that key-value pair is retrieved and its value is sent to the data collector. If not, l_u^t is freshly perturbed with LDP to obtain $\tilde{l}_u^t \leftarrow \Psi(l_u^t)$. This $\tilde{l}_u^t$ is sent to the data collector; furthermore, the pair $\langle l_u^t, \tilde{l}_u^t \rangle$ is inserted to the cache so that if l_u^t is observed again in future timestamps, its perturbed counterpart will be retrieved from the cache.

Algorithm 3 Replay Defense

Input: User u's real locations $l_u^1, l_u^2, ..., l_u^n$

Output: User u's reported values $\tilde{l}_u^1, \tilde{l}_u^2, ..., \tilde{l}_u^n$

$cache \leftarrow$ Initialize as empty key-value store
for $t = 1$ to n **do**
  if an entry with key $= l_u^t$ is found in $cache$ **then**
    $\langle l_u^t, val \rangle \leftarrow cache.retrieve(l_u^t)$
    $\tilde{l}_u^t \leftarrow val$
  else
    $\tilde{l}_u^t \leftarrow \Psi(l_u^t)$
    $cache.insert(\langle l_u^t, \tilde{l}_u^t \rangle)$
  end if
end for
return $\tilde{l}_u^1, \tilde{l}_u^2, ..., \tilde{l}_u^n$

Since a perturbed value is not perturbed again, the replay defense does not suffer from the increased utility loss in memoization. Furthermore, it can serve as an effective defense against longitudinal attacks when the same location is repeated multiple times, e.g., attacks against stationary users, or users with limited mobility, or users with frequently repeating locations (such as home, workplace, etc.). For such users, the benefit of the replay defense is that the attacker is no longer able to observe multiple perturbed instances of the same underlying location, but rather, a single perturbed instance. Therefore, the attacker's PA becomes upper bounded by the PA that can be achieved with only a single perturbed instance. On the other hand, the main disadvantage of the replay defense is the size of the cache. Over time, as the user visits many different locations, the cache on the user's device will keep growing. This implies that the secure storage space requirement on the user's device will keep increasing. Users with higher mobility (i.e., many different non-repeating locations) will suffer from this problem more.

Algorithm 4 Replication Defense

Input: User u's real locations $l_u^1, l_u^2, ..., l_u^n$

Output: User u's reported values $\tilde{l}_u^1, \tilde{l}_u^2, ..., \tilde{l}_u^n$

1: $\tilde{l}_u^1 \leftarrow \Psi(l_u^1)$
2: **for** $t = 2$ to n **do**
3: **if** $l_u^t = l_u^{t-1}$ **then**
4: Retrieve $\tilde{l}_u^{t-1}$ and set $\tilde{l}_u^t \leftarrow \tilde{l}_u^{t-1}$
5: **else**
6: $\tilde{l}_u^t \leftarrow \Psi(l_u^t)$
7: **end if**
8: **end for**
9: **return** $\tilde{l}_u^1, \tilde{l}_u^2, ..., \tilde{l}_u^n$

6.3 Replication

Our third defense is replication, which aims to circumvent the increasing cache size problem of replay and also the double perturbation problem of memoization. In replication, if $l_u^t = l_u^{t-1}$, then the user replicates the perturbed value from the previous timestamp in the current timestamp, i.e., $\tilde{l}_u^t \leftarrow \tilde{l}_u^{t-1}$ and $\tilde{l}_u^t$ is reported to the data collector. If $l_u^t \neq l_u^{t-1}$, then the user freshly perturbs the current location and reports it to the data collector, i.e., $\tilde{l}_u^t \leftarrow \Psi(l_u^t)$. Since the user compares the current location l_u^t with only the previous location l_u^{t-1}, there is no need to store a cache; only the immediately previous pair $\langle l_u^{t-1}, \tilde{l}_u^{t-1} \rangle$ needs to be stored. Also, since $\tilde{l}_u^{t-1}$ is replicated without a second step of perturbation, memoization's double perturbation problem does not apply.

The pseudocode of our replication defense is shown in Algorithm 4. The first location is perturbed $\tilde{l}_u^1 \leftarrow \Psi(l_u^1)$ and the result is stored in memory as a pair: $\langle l_u^1, \tilde{l}_u^1 \rangle$. Then, for next timestamps $t = 2$ to n, the current location l_u^t is compared with the immediately previous location l_u^{t-1}. If $l_u^t = l_u^{t-1}$, then the perturbed value from the previous timestamp $\tilde{l}_u^{t-1}$ is retrieved and assigned as the perturbed value in the current timestamp, i.e., $\tilde{l}_u^t \leftarrow \tilde{l}_u^{t-1}$. This means that as long as the user's

location remains the same as before, the same perturbed value will be sent to the data collector, i.e., replicated. However, if $l_u^t \neq l_u^{t-1}$, then l_u^t is freshly perturbed and the result is sent to the data collector: $\tilde{l}_u^t \leftarrow \Psi(l_u^t)$.

The replication defense only needs to store the immediately previous location l_u^{t-1} and its perturbed counterpart $\tilde{l}_u^{t-1}$. Therefore, its storage requirement is similar to memoization and much smaller than replay. Furthermore, since each value reported from the user to the data collector is the result of a single perturbation, it does not suffer from the utility loss of memoization. On the other hand, replication is a weaker defense in terms of the protection it offers against our attacks. In particular, since replication does not store a *cache*, it cannot address the cases where the same location sequence is repeated at different times. For example, consider Alice who is stationary at home at $t = 1, 2, 3$. Replication would cause Alice to report $\tilde{l}_u^1 = \tilde{l}_u^2 = \tilde{l}_u^3$, which means that its protection for the first 3 timestamps would be equivalent to replay. Then, say that Alice visits different locations between $t = 4$ and 10. Finally, Alice arrives back home and she is stationary at home at $t = 11, 12, 13$. Alice would report $\tilde{l}_u^{11} = \tilde{l}_u^3$ in replay, but she can report $\tilde{l}_u^{11} \neq \tilde{l}_u^3$ in replication. Hence, replication does not prevent the attacker from obtaining multiple perturbed observations of the same underlying location if that location is occurring at different times intermittently. As a result, replication is a weaker defense than replay.

6.4 Experiment Results and Discussion

We use the same experiment setup and protocols from the previous chapters to evaluate the defenses proposed in this chapter. We explore the effectiveness of the defenses on multiple protocols and datasets, and we discuss their PA and efficiency implications.

6.4.1 Experiments with Stationary Users

For stationary users, we used the same experiment setup as in Section 4.4.2. The results for all three datasets, four protocols, and three defenses (plus the case without any defense, denoted by "No Defense") are shown in Figure 6.1. We make several

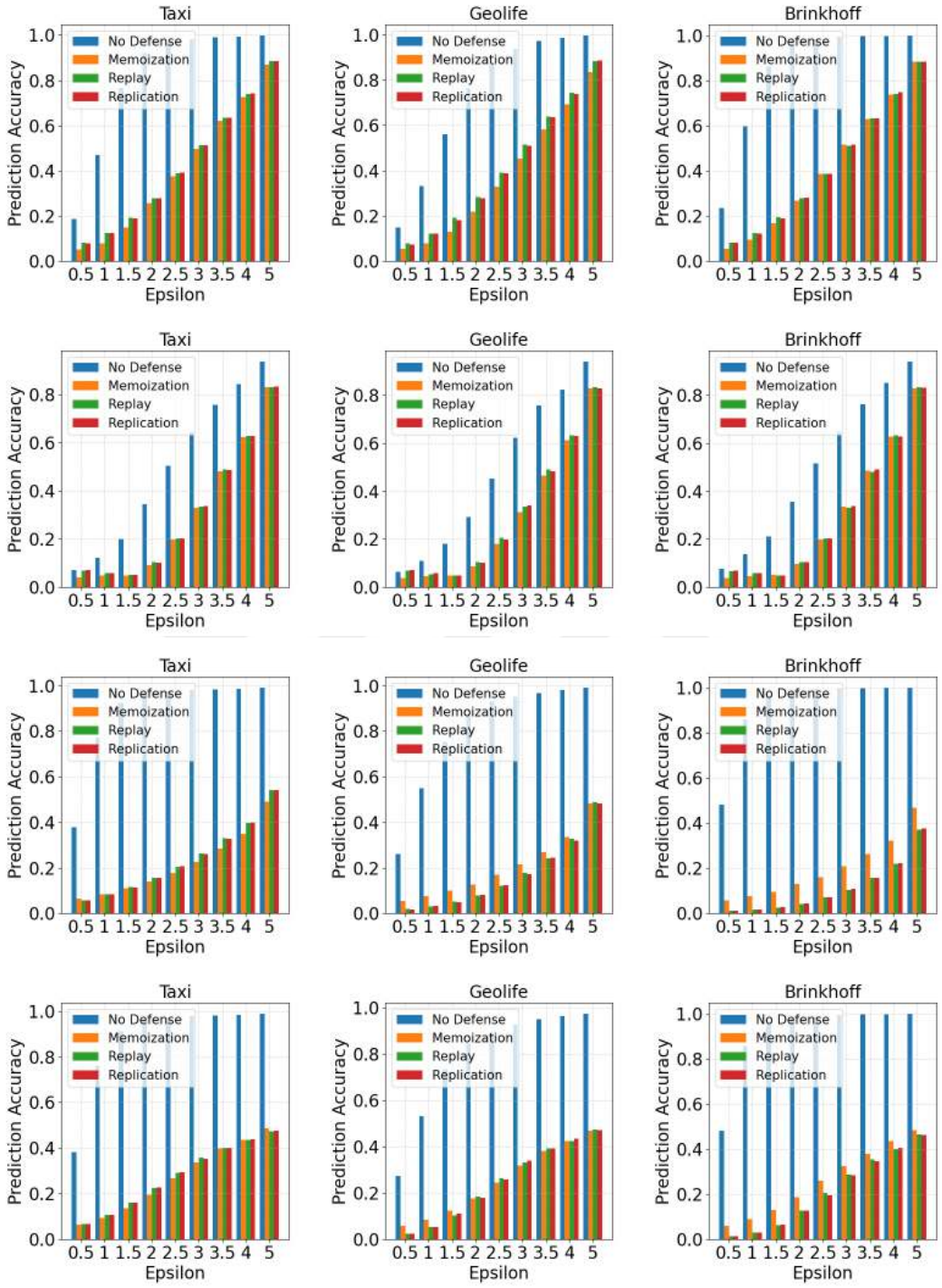

Figure 6.1: Prediction accuracies of our attacks on stationary users, with and without the defenses. First row is with the GRR protocol, second row is OLH, third row is RAPPOR, fourth row is OUE.

observations from Figure 6.1. First, we observe that the PA benefits of the defenses are most pronounced when ε is not too small or too large, e.g., when ε is between 1.5 and 4. This is because when ε is small, PAs of the original attack are not high, and although the defenses yield some PA reduction, the reductions are usually not that significant. When ε is large, again the defenses yield some PA reduction, but the PAs still remain high. Thus, medium ε values between 1.5 and 4, which are also commonly used ε values in the LDP literature and deployments, are when the defenses are most impactful.

Second, we observe that the defenses are more effective on RAPPOR and OUE protocols compared to GRR and OLH. Recall that RAPPOR and OUE are bitvector-based protocols, and the users' perturbed values are one of $2^{|\mathcal{G}|}$ possible outcomes. In contrast, the number of outcomes in GRR and OLH are $\leq |\mathcal{G}|$. From the practical perspective of the attack, the increased number of possible outcomes increases the difficulty of inferring the user's true location, which arises from the size of the emission probability matrix $\mathcal{B}$. From the perspective of the defenses, the large output space of the protocol yields a large output space for the defense as well. For example, after memoization performs perturbation twice, the resulting bitvector will be one of the $2^{|\mathcal{G}|}$ possible bitvectors. In contrast, the number of outcomes in GRR and OLH is much smaller. Consequently, when a defense is applied, the likelihood of the observed output to be substantially different from what the output would have been without the defense is high. Hence, all defenses are relatively more effective on RAPPOR and OUE.

Third, we observe that replay and replication perform similarly in general. This is an intuitive result, since these experiments assume stationary users. When the users are stationary, the $\tilde{l}_u^1, \tilde{l}_u^2, ..., \tilde{l}_u^n$ produced by replay and replication will be identical in expectation. This is because the same $\tilde{l}_u^i$ will be read from the cache in replay, and the same $\tilde{l}_u^i$ will be produced for all $i \in [1, n]$ in case of replication. The results in Figure 6.1 validate the expectation that replay and replication should perform similarly, as they yield similar PAs. In contrast, memoization works differently from replay and replication, and therefore it yields different PAs. Memoization yields better results especially for small ε values and datasets with shorter trajectories,

such as Taxi and Geolife. Yet, on the Brinkhoff dataset (which contains longer trajectories) and with larger ε values, PAs of memoization are higher than PAs of replay and replication. These results yield important insights into which defense should be preferred under what conditions.

Finally, we observe that the defenses are less effective in case of GRR and OLH protocols compared to RAPPOR and OUE. The reason behind this observation is that this is a stationary setup and large ε yields high PAs in a stationary setup. In many cases, few observations (i.e., $n = 1$ or 2) are sufficient for the attacker to make a correct prediction. In such cases, the defenses do not yield much benefit. For example, replay and replication defenses would be reporting $\tilde{l}_u^1 = \tilde{l}_u^2 = ... = \tilde{l}_u^n$, but consider that even $\tilde{l}_u^1$ is sufficient for the attacker to make a correct prediction. The occurrence probability of this is larger for GRR and OLH protocols since their output space is smaller (as discussed above), whereas it is smaller for RAPPOR and OUE protocols since their output space is of size $2^{|\mathcal{G}|}$. This is why the defenses are less effective under GRR and OLH with large ε.

6.4.2 Experiments with Mobile Users

For mobile users, we used the same experiment setup as in Section 4.4.3. The results for all three datasets, four protocols, and three defenses plus "No Defense" are shown in Figure 6.2. First, similar to Figure 6.1, we observe that PAs increase as ε increases, and each defense reduces the PAs, often by a nontrivial amount. Different datasets exhibit different PAs, but the aforementioned trends hold. Nevertheless, in general, employing any one of the proposed defenses will reduce the attacker's PA.

Second, we observe from the results that our memoization defense achieves the highest reductions in PA across all datasets and protocols. The reductions achieved by memoization are substantially larger than replay and replication. So, while memoization incurs higher utility loss compared to replay and replication, in cases where PA reduction is the primary goal, memoization can be preferred.

Third, we observe that while replay and replication yield noticeable PA reductions for RAPPOR and OUE protocols, their PA reductions for GRR and OLH are

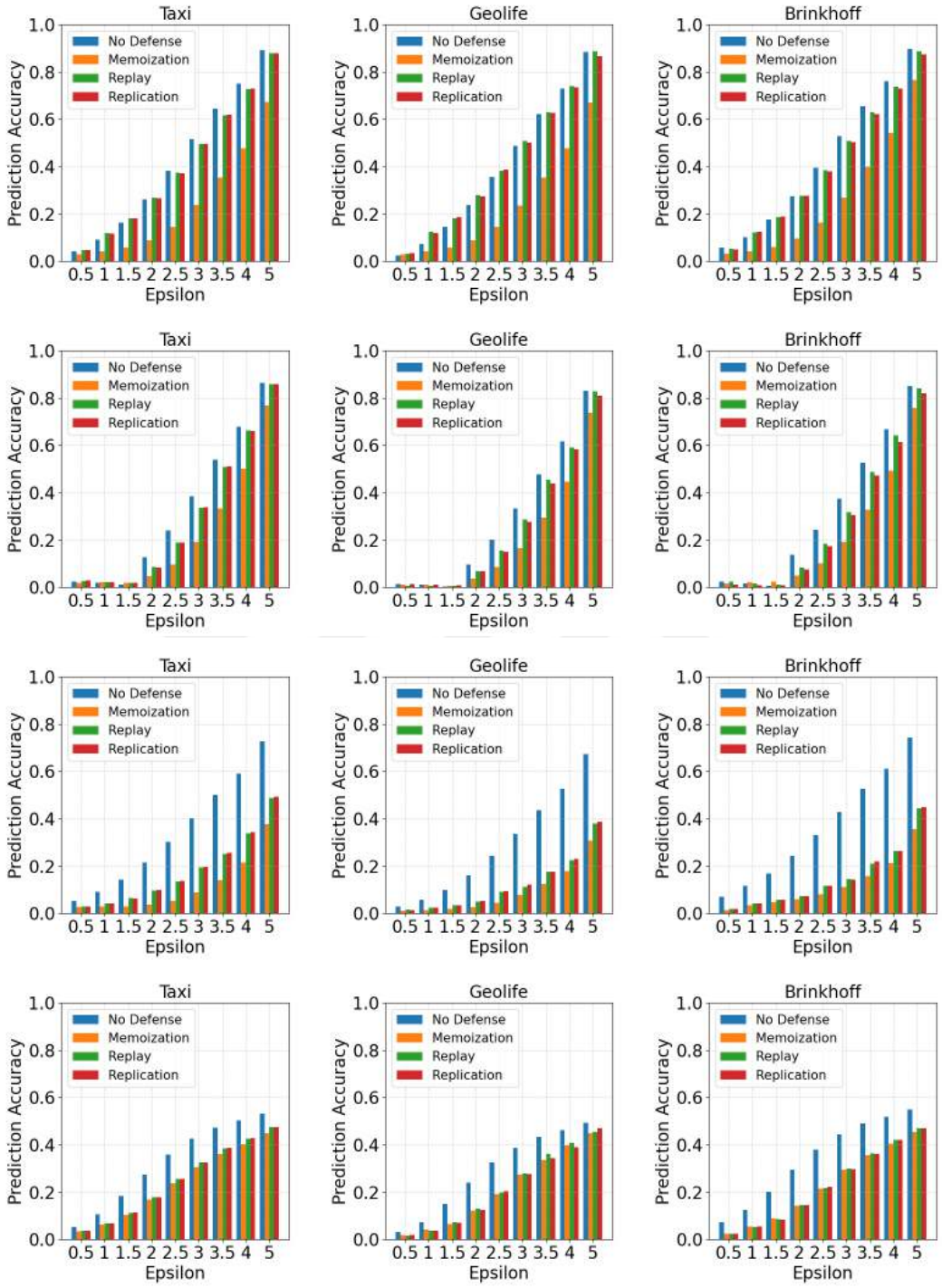

Figure 6.2: Prediction accuracies of our attacks on mobile users, with and without the defenses. First row is with the GRR protocol, second row is OLH, third row is RAPPOR, fourth row is OUE.

not that high. This can be attributed to the mobile nature of the users. Consider a user who is highly mobile, e.g., $l_u^1 \neq l_u^2 \neq ... \neq l_u^n$. In this case, since the same location is never repeated, both replay and replication defenses would freshly perturb each l_u^i for $i \in [1, n]$. That is, the replay defense would never retrieve any values from its cache, and the replication defense would never retrieve $\tilde{l}_u^{t-1}$ and set $\tilde{l}_u^t \leftarrow \tilde{l}_u^{t-1}$. Consequently, the defenses do not provide strong protection. In general, the protection of replay and replication defenses increase as more and more locations are repeated (duplicated) in $l_u^1, l_u^2, ..., l_u^n$. Hence, increased mobility amounts decrease the effectiveness of replay and replication.

6.4.3 Summary and Recommendations

In summary, the proposed defenses generally reduce attackers' PA for both stationary and mobile users, therefore they are beneficial to defend against the attacks. Replay and replication defenses are more effective for bitvector-based protocols largely due to the structure of the emission probability matrix; however, they are not as effective for protocols such as GRR and OLH. Memoization generally yields good reductions in PA across varying datasets, protocols, and ε. On the other hand, replay and replication perform a single round of perturbation whereas memoization performs two perturbations. Hence, the higher PA reduction achieved by memoization comes at the cost of increased utility loss.

Considering these trade-offs, different defenses can be preferable under different factors and conditions. Based on our experiments and analyses, we can provide the following recommendations.

If users are stationary or near-stationary:

- Replay and replication perform similarly, both in expectation and experimentally. Consequently, they yield similar PAs.

- If trajectories are short and ε is small, then memoization can be preferred due to its low PA.

- If trajectories are long and/or ε is large, then replay or replication should be

preferred. Among the two, replication is better in terms of simplicity and lower memory usage (does not need to store a cache). However, if there is no memory constraint and the user is near-stationary but perhaps not fully stationary, then replay should be preferred since it is a stronger defense.

If users are mobile:

- Memoization can yield much higher PA reductions; therefore, we recommend the use of memoization. However, since memoization performs perturbation twice, it causes higher utility loss compared to replay and replication.

- If utility loss is important and therefore memoization is not preferred, then one of replay or replication can be used. While replay is stronger than replication intuitively, based on the experiment results, their PAs are similar. If there is a memory constraint, then replication can be preferred, otherwise replay should be preferred.

Chapter 7

CONCLUSION

In this thesis, we proposed inference attacks against continuous location sharing under LDP. Our attacks were categorized into two main types: (i) statistical Bayesian attacks targeting stationary users, and (ii) HMM-based attacks focusing on mobile users. For HMM-based attacks, "chain" and "informed" attack extensions were proposed, which were experimentally shown to improve attack effectiveness. Furthermore, we showed that both attacks are better fit for the scenario that they were designed for, i.e., statistical Bayesian attacks achieve higher PA against stationary users whereas HMM-based attacks achieve higher PA against mobile users.

Motivated by the high memory usage and execution times of our HMM-based attacks on bitvector-based protocols such as RAPPOR and OUE, we then proposed heuristic methods to improve attack efficiency. By removing observable states with low occurrence probabilities from the HMMs, we showed that more than 20-fold reductions in execution time and memory usage can be attained with negligible negative impact on attack effectiveness.

Lastly, we proposed three defenses to counter the threats posed by our inference attacks: memoization, replay, and replication. We evaluated the three defenses in terms of the reduction in attacks' PA as well as their efficiency and utility implications. Experimental results showed that the defenses are beneficial in defending against the attacks, as they generally reduce attackers' PAs for both stationary and mobile users. We also provided recommendations regarding when to use which defense, considering the trade-offs in PA reduction, efficiency, and utility.

There are several directions for future work. First, our attacks can be executed on differentially private stream processing methods and/or w-event based privacy protection methods. In these contexts, our inference attacks can serve as tools for practical privacy measurement, similar to how membership inference or property

inference attacks are used in machine learning. Second, applications of our attacks on other continuous data types such as sensor readings or power consumption data can be studied. Third, new LDP protocols which specifically target continuous location sharing can be developed. Considering that different protocols have different resilience against our attacks (e.g., OLH, which uses hashing, yields lower PA and higher NDE); the resilient aspects of different protocols (e.g., hashing) can be combined to design protocols that are more resilient against our attacks.

BIBLIOGRAPHY

[app, 2020] (2020). Apple – learning with privacy at scale. `https://machinelearning.apple.com/docs/learning-with-privacy-at-scale/appledifferentialprivacysystem.pdf`.

[Alptekin and Gursoy, 2023] Alptekin, E. and Gursoy, M. E. (2023). Building quadtrees for spatial data under local differential privacy. In *IFIP Annual Conference on Data and Applications Security and Privacy*, pages 22–39. Springer.

[Arcolezi et al., 2022] Arcolezi, H. H., Couchot, J.-F., Al Bouna, B., and Xiao, X. (2022). Improving the utility of locally differentially private protocols for longitudinal and multidimensional frequency estimates. *Digital Communications and Networks*.

[Arcolezi et al., 2023] Arcolezi, H. H., Gambs, S., Couchot, J.-F., and Palamidessi, C. (2023). On the risks of collecting multidimensional data under local differential privacy. *Proceedings of the VLDB Endowment (PVLDB)*, 16(5):1126–1139.

[Balioglu et al., 2024] Balioglu, B. K., Khodaie, A., Taweel, A., and Gursoy, M. E. (2024). Grid-based decompositions for spatial data under local differential privacy. *arXiv preprint arXiv:2407.21624*.

[Baum and Petrie, 1966] Baum, L. E. and Petrie, T. (1966). Statistical inference for probabilistic functions of finite state markov chains. *The Annals of Mathematical Statistics*, 37(6):1554–1563.

[Brinkhoff, 2002] Brinkhoff, T. (2002). A framework for generating network-based moving objects. *GeoInformatica*, 6(2):153–180.

[Cao et al., 2021] Cao, X., Jia, J., and Gong, N. Z. (2021). Data poisoning attacks to local differential privacy protocols. In *30th USENIX Security Symposium*, pages 947–964.

[Chen et al., 2016] Chen, R., Li, H., Qin, A., Kasiviswanathan, S. P., and Jin, H. (2016). Private spatial data aggregation in the local setting. In *2016 IEEE 32nd International Conference on Data Engineering (ICDE)*, pages 289–300. IEEE.

[Cheu et al., 2021] Cheu, A., Smith, A., and Ullman, J. (2021). Manipulation attacks in local differential privacy. In *2021 IEEE Symposium on Security and Privacy (SP)*, pages 883–900. IEEE.

[Cormode et al., 2018] Cormode, G., Jha, S., Kulkarni, T., Li, N., Srivastava, D., and Wang, T. (2018). Privacy at scale: Local differential privacy in practice. In *Proceedings of the 2018 International Conference on Management of Data*, pages 1655–1658. ACM.

[Cunningham et al., 2021] Cunningham, T., Cormode, G., Ferhatosmanoglu, H., and Srivastava, D. (2021). Real-world trajectory sharing with local differential privacy. *Proceedings of the VLDB Endowment*, 14(11):2283–2295.

[Ding et al., 2017] Ding, B., Kulkarni, J., and Yekhanin, S. (2017). Collecting telemetry data privately. In *Advances in Neural Information Processing Systems*, pages 3571–3580.

[Du et al., 2023] Du, Y., Hu, Y., Zhang, Z., Fang, Z., Chen, L., Zheng, B., and Gao, Y. (2023). Ldptrace: Locally differentially private trajectory synthesis. *Proceedings of the VLDB Endowment*, 16(8):1897–1909.

[Erlingsson et al., 2014] Erlingsson, Ú., Pihur, V., and Korolova, A. (2014). Rappor: Randomized aggregatable privacy-preserving ordinal response. In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, pages 1054–1067. ACM.

[Fanti et al., 2016] Fanti, G., Pihur, V., and Erlingsson, Ú. (2016). Building a rappor with the unknown: Privacy-preserving learning of associations and data dictionaries. *Proceedings on Privacy Enhancing Technologies*, 2016(3):41–61.

[Forney, 1973] Forney, G. D. (1973). The viterbi algorithm. *Proceedings of the IEEE*, 61(3):268–278.

[Gadotti et al., 2022] Gadotti, A., Houssiau, F., Annamalai, M. S. M. S., and de Montjoye, Y.-A. (2022). Pool inference attacks on local differential privacy: Quantifying the privacy guarantees of apple's count mean sketch in practice. In *31st USENIX Security Symposium*, pages 501–518.

[Gursoy, 2024] Gursoy, M. E. (2024). Longitudinal attacks against iterative data collection with local differential privacy. In *Turkish Journal of Electrical Engineering and Computer Sciences*, pages 198–218. Tubitak.

[Gursoy et al., 2022] Gursoy, M. E., Liu, L., Chow, K.-H., Truex, S., and Wei, W. (2022). An adversarial approach to protocol analysis and selection in local differential privacy. *IEEE Transactions on Information Forensics and Security*, 17:1785–1799.

[Hong et al., 2021] Hong, D., Jung, W., and Shim, K. (2021). Collecting geospatial data with local differential privacy for personalized services. In *2021 IEEE 37th International Conference on Data Engineering (ICDE)*, pages 2237–2242. IEEE.

[Hong et al., 2022] Hong, D., Jung, W., and Shim, K. (2022). Collecting geospatial data under local differential privacy with improving frequency estimation. *IEEE Transactions on Knowledge and Data Engineering*.

[Huang et al., 2024] Huang, K., Ouyang, G., Ye, Q., Hu, H., Zheng, B., Zhao, X., Zhang, R., and Zhou, X. (2024). Ldpguard: Defenses against data poisoning attacks to local differential privacy protocols. *IEEE Transactions on Knowledge and Data Engineering*.

[Imola et al., 2022] Imola, J., Chowdhury, A. R., and Chaudhuri, K. (2022). Robustness of locally differentially private graph analysis against poisoning. *arXiv preprint arXiv:2210.14376*.

[Kairouz et al., 2016] Kairouz, P., Bonawitz, K., and Ramage, D. (2016). Discrete distribution estimation under local privacy. In *Proceedings of the 33rd International Conference on International Conference on Machine Learning - Volume 48*, ICML'16, page 2436–2444. JMLR.org.

[Kato et al., 2021] Kato, F., Cao, Y., and Yoshikawa, M. (2021). Preventing manipulation attack in local differential privacy using verifiable randomization mechanism. In *35th Annual IFIP WG 11.3 Conference on Data and Applications Security and Privacy (DBSec)*, pages 43–60. Springer.

[Kim et al., 2018] Kim, J. W., Kim, D.-H., and Jang, B. (2018). Application of local differential privacy to collection of indoor positioning data. *IEEE Access*, 6:4276–4286.

[Li et al., 2023] Li, X., Li, N., Sun, W., Gong, N. Z., and Li, H. (2023). Fine-grained poisoning attack to local differential privacy protocols for mean and variance estimation. In *32nd USENIX Security Symposium (USENIX Security 23)*, pages 1739–1756.

[Li et al., 2024] Li, X., Li, Z., Li, N., and Sun, W. (2024). On the robustness of ldp protocols for numerical attributes under data poisoning attacks. *arXiv preprint arXiv:2403.19510*.

[Moreira-Matias et al., 2013] Moreira-Matias, L., Gama, J., Ferreira, M., Mendes-Moreira, J., and Damas, L. (2013). Predicting taxi–passenger demand using streaming data. *IEEE Transactions on Intelligent Transportation Systems*, 14(3):1393–1402.

[Murakami and Takahashi, 2021] Murakami, T. and Takahashi, K. (2021). Toward

evaluating re-identification risks in the local privacy model. *Transactions on Data Privacy*, 14(3):79–116.

[Navidan et al., 2022] Navidan, H., Moghtadaiee, V., Nazaran, N., and Alishahi, M. (2022). Hide me behind the noise: Local differential privacy for indoor location privacy. In *2022 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW)*, pages 514–523. IEEE.

[Rabiner and Juang, 1986] Rabiner, L. and Juang, B. (1986). An introduction to hidden markov models. *IEEE ASSP Magazine*, 3(1):4–16.

[Rabiner, 1989] Rabiner, L. R. (1989). A tutorial on hidden markov models and selected applications in speech recognition. *Proceedings of the IEEE*, 77(2):257–286.

[Song et al., 2023] Song, S., Xu, L., and Zhu, L. (2023). Efficient defenses against output poisoning attacks on local differential privacy. *IEEE Transactions on Information Forensics and Security*, 18:5506–5521.

[Tire and Gursoy, 2024] Tire, E. and Gursoy, M. E. (2024). Answering spatial density queries under local differential privacy. *IEEE Internet of Things Journal*.

[Viterbi, 1967] Viterbi, A. (1967). Error bounds for convolutional codes and an asymptotically optimum decoding algorithm. *IEEE Transactions on Information Theory*, 13(2):260–269.

[Wang et al., 2022] Wang, H., Hong, H., Xiong, L., Qin, Z., and Hong, Y. (2022). L-srr: Local differential privacy for location-based services with staircase randomized response. In *Proceedings of the 2022 ACM SIGSAC Conference on Computer and Communications Security*, page 2809–2823, New York, NY, USA. Association for Computing Machinery.

[Wang et al., 2017] Wang, T., Blocki, J., Li, N., and Jha, S. (2017). Locally differentially private protocols for frequency estimation. In *Proc. of the 26th USENIX Security Symposium*, pages 729–745.

[Warner, 1965] Warner, S. L. (1965). Randomized response: A survey technique for eliminating evasive answer bias. *Journal of the American Statistical Association*, 60(309):63–69.

[Wu et al., 2022] Wu, Y., Cao, X., Jia, J., and Gong, N. Z. (2022). Poisoning attacks to local differential privacy protocols for key-value data. In *31st USENIX Security Symposium*, pages 519–536.

[Xiong et al., 2020] Xiong, X., Liu, S., Li, D., Cai, Z., and Niu, X. (2020). A comprehensive survey on local differential privacy. *Security and Communication Networks*, 2020:1–29.

[Yang et al., 2022] Yang, J., Cheng, X., Su, S., Sun, H., and Chen, C. (2022). Collecting individual trajectories under local differential privacy. In *2022 23rd IEEE International Conference on Mobile Data Management (MDM)*, pages 99–108. IEEE.

[Yang et al., 2020] Yang, M., Lyu, L., Zhao, J., Zhu, T., and Lam, K.-Y. (2020). Local differential privacy and its applications: A comprehensive survey. *arXiv preprint arXiv:2008.03686*.

[Zheng et al., 2010] Zheng, Y., Xie, X., and Ma, W.-Y. (2010). Geolife: A collaborative social networking service among user, location and trajectory. *IEEE Data Eng. Bull.*, 33:32–39.

[Zheng et al., 2024] Zheng, Z., Li, Z., Huang, C., Long, S., Li, M., and Shen, X. (2024). Data poisoning attacks and defenses to ldp-based privacy-preserving crowdsensing. *IEEE Transactions on Dependable and Secure Computing*.