Privacy-disclosure and Privacy-Preserving for Online Data

by Tong Yan

B.S. in Computer Science, Jun 2010, Sichuan Normal University
M.S. in Computer Science, May 2014, The George Washington University

A Dissertation submitted to

The Faculty of
The School of Engineering and Applied Science
of The George Washington University
in partial satisfaction of the requirements
for the degree of Doctor of Philosophy

August 31, 2019

Dissertation directed by

Xiuzhen Cheng
Professor of Computer Science

Nan Zhang
Professor of Info Technology & Analytics
The School of Engineering and Applied Science of The George Washington University certifies that Tong Yan has passed the Final Examination for the degree of Doctor of Philosophy as of May 10, 2019. This is the final and approved form of the dissertation.

Privacy-disclosure and Privacy-Preserving for Online Data

Tong Yan

Dissertation Research Committee:

Xiu Zheng Cheng, Professor of Computer Science,
Dissertation Director

Hyeong-Ah Choi, Professor of Computer Science,
Committee Member

Nan Zhang, Professor of Info Technology & Analytics,
American University, Committee Member

Howie Huang, Electrical & Computer Engineering,
Committee Member
Dedication

To my family.
Abstract

Privacy-disclosure and Privacy-Preserving for Online Data

There are a number of web databases existed in the world. Some of which are hidden from the public view. They provide search interfaces to users that allow issuing searching queries and present information based on the user request. The information is usually ordered searching results (e.g. return $top - k$ results to users, where $k$ is much smaller than the size of database) from proper ranking functions. For example, Amazon.com, Expedia.com, etc. Each site dynamically creates pages based on the user request. Some other web databases, however, are public to users that allow users to completely retrieve the dataset. For example, credit card history transactions, medical data for public health, etc. Each database grants access to users to retrieve the whole dataset.

In this dissertation, we consider two kinds of problem. The first is to infer the hidden information over web databases. One of which is to use the published time-series data to infer a user’s daily activities. Theoretical analysis and real-world experiments demonstrate the effectiveness of our proposed algorithms over the baseline algorithm. We also investigate a novel problem on the implication of the information asymmetry model with transparency strategies. We propose IHF-matching Algorithm and the real-world experiments demonstrate the high success inference rate over real datasets. The second is to protect privacy in web databases. In this dissertation, we propose a novel privacy-preserving framework. Our framework protects private attributes’ privacy not only under inference attacks but also under arbitrary attack methods. We demonstrate the effectiveness and efficiency of our framework through theoretical analysis, and extensive experiments over real-world datasets.
# Table of Contents

Dedication ........................................................................ iv

Abstract ........................................................................... v

List of Figures ..................................................................... viii

List of Tables ...................................................................... ix

1 Introduction ..................................................................... 1

1.1 Web Databases ................................................................. 1

1.2 Privacy Disclosure from Time-series Data ......................... 1

1.2.1 Outline of Technical Results ........................................ 4

1.2.2 Summary of Contributions .......................................... 4

1.3 Privacy Implications of Hotel Bookings ............................ 5

1.3.1 Outline of Technical Results ........................................ 7

1.3.2 Summary of Contributions .......................................... 7

1.4 Privacy-preserving Framework ....................................... 7

1.4.1 Outline of Technical Results ........................................ 8

1.4.2 Summary of Contributions .......................................... 9

2 Privacy Disclosure from Wearable Devices ......................... 10

2.1 Background ................................................................. 10

2.1.1 Time Series ............................................................ 11

2.1.2 Euclidean Distance .................................................. 12

2.2 Problem Definition ........................................................ 12

2.3 Baseline Algorithms for Event Detection from Time-Series Data .................................................. 13

2.3.1 Approximate Matching algorithm .............................. 13

2.3.2 Vertical Shift Approximate Matching ......................... 14

2.4 Implementation of Inference Attacks ............................... 15

3 Privacy Implications of Hotel Bookings .............................. 17

3.1 Background ................................................................. 17

3.2 Preliminaries ............................................................... 20

3.2.1 Asymmetry model with transparency strategies .......... 20

3.2.2 Matching Algorithm ............................................... 21

3.3 Problem Definition ........................................................ 22

3.4 Baseline Inference and IHF-matching Algorithm ............ 23

3.4.1 Baseline Inference Algorithm ................................. 23

3.4.2 IHF-matching algorithm ........................................ 23

4 Privacy-preserving Framework ....................................... 25

4.1 Motivation ................................................................. 25

4.2 Privacy-preserving Framework .................................... 28
# List of Figures

1.1 A linear representation of a pedometer sensed data ............................................. 3  
1.2 Demonstrate of a hot deal from Hotwire.com ..................................................... 5  
1.3 Demonstrate of an attack over Hotwire.com ..................................................... 6  

2.1 A pedometer sensed data .................................................................................... 11  
2.2 $X$ is shifted down to get $Y$ .............................................................................. 14  

3.1 Priceline’s Name Your Own Price ....................................................................... 19  
3.2 Hotwire’s Hot Rate Hotels .................................................................................. 19  

4.1 A Hamiltonian Circuit in B .................................................................................. 41  

5.1 Number of errors when detect event (a) in 20 tests .......................................... 46  
5.2 Number of errors when detect event (b) in 20 tests .......................................... 46  
5.3 Number of errors when detect event (c) in 20 tests .......................................... 47  
5.4 1 reading per minute ......................................................................................... 49  
5.5 1 reading per 2 minutes ....................................................................................... 49  
5.6 1 reading per 3 minutes ....................................................................................... 50  
5.7 Inference Results of Hotels in Top-10 popular cities of the United States ......... 53  
5.8 Inference Success Rate of Hotels in Top-10 popular cities of the United States  54  
5.9 Attack Guess Success Rate ................................................................................ 57  
5.10 Virtual tuples (True tuples) vs random .............................................................. 58  
5.11 Utility loss of virtual tuples ............................................................................... 58  
5.12 Utility loss of true tuples ................................................................................... 58
List of Tables

4.1 Vertex matrix .......................................................... 42
Chapter 1: Introduction

1.1 Web Databases

There are a number of web databases existed in the world. Some of which are hidden from the public view. They provide search interfaces to users that allow issuing searching queries and present information based on the user request\[1\]. The information is usually ordered searching results (e.g. return top $k$ results to users, where $k$ is much smaller than the size of database) from proper ranking functions. For example, Amazon.com, Expedia.com, etc. Each site dynamically creates pages based on the user request. Some other web databases, however, are public to users that allow users to completely retrieve the dataset. For example, credit card history transactions, medical data for public health, etc. Each database grants access to users to retrieve the whole dataset.

1.2 Privacy Disclosure from Time-series Data

Problem Motivation: For time-series datasets, in this paper we consider that a third party can use the published time-series data to infer a user’s daily activities. The reason that we choose to study the inference of private information from wearable device outputs is two-fold:

- First is the wide (mis)perception that no sensitive information can be leaked from pedometer outputs. This is evidenced by the fact that many popular wearable platforms (e.g., Fitbit Dashboard, UP by Jawbone) build the core of their social features on the sharing of pedometer outputs - allowing users to race with each other (in terms of the number of steps walked), tracking each other’s progress, etc [2, 3]. Some of these platforms (e.g., Fitbit Dashboard) even adopt an opt-out model for the social features - i.e., step counts are shared by default unless the user switches it off. If a
user’s behavior can be easily inferred from the pedometer outputs, then the sharing of such data may lead to significant privacy disclosures unanticipated by a user of wearable devices.

- The second reason that we choose to study this topic is that indeed a well-understood correlation between the “behavior” of a person (e.g., grocery shopping or visiting a coffee shop) and the actions (e.g., walking, running, climbing stairs) sensed by a pedometer in the field of human behavioral science. Specifically, studies on human mobility [4, 5] introduced the concept of reproducible patterns, i.e., a sequence of (sensed) actions, to denote an instinctive behavioral sequence such as Walking Stride Length and Stride Frequency. Given such an understanding, the objective of this paper is to study whether the pedometer data provided by off-the-shelf wearable devices (through their data sharing mechanisms such as APIs) can reveal specific behavioral sequences of the user.

**Challenges:** It is important to understand the limitations of existing safe-guarding techniques used in practice - specifically, the potential inference of sensitive information from seemingly insensitive sensor outputs. In particular, we study in this paper a novel problem of whether one can infer the “behavior” of a user (precise definition of “behavior” to follow later in the paper) - e.g., moving from one place to another, visiting a coffee shop, grocery shopping, etc. - from pedometer outputs. Pedometer is a sensor that counts the number of steps taken by a user. It can be implemented with a simple mechanical switch [6] or a more complex MEMS inertial sensor which detects acceleration on all three axes for more precise counting [7, 8]. Today, a state-of-the-practice pedometer can achieve an accuracy level of within 5% [9].

An important issue in the processing of time-series data is the data representation, which is crucial for determining the similarity between time-series sequences. Examples of popular data representations include Fourier Transforms, Wavelets and Linear Representation, etc., with the two former ones (i.e., Fourier and wavelet transformations) often applied over
sequences that are locally stationary in time [10, 11]. Since our study in the paper focuses on detecting from a user’s daily activities an event that is unlikely to repeat multiple times during the course of a day, we adopt the basic linear representation to represent time-series data in the paper. An example of such a linear representation of a pedometer sensed data is illustrated in Figure 1.1.

![Linear Representation](image.png)

**Figure 1.1: A linear representation of a pedometer sensed data**

**Problem of Privacy Disclosure from Wearable Devices:** In this paper, we study the inference attack to time-series data. Our main focus in the paper is to examine whether a pre-defined event, e.g., walking from office to a nearby coffee shop, can be identified from amongst many other routine or random activities undertaken by the user and captured by the wearable device. To properly determine whether an event of interest has occurred within a sequence of measured user activities, it is important to measure the distance (or similarity) between two time-series (sub-)sequences. In this paper, we introduce the *Euclidean distance*. It is arguably the most common distance metric used in the literature.
1.2.1 Outline of Technical Results

Firstly, instead of directly using the pedometer readings, we transfer the pedometer readings to time-series tracking-measure sequences. We demonstrate through real-world experiments that surprisingly it is often possible to infer such behavior with a high success probability, raising privacy concerns on the sharing of such information as currently supported by various wearable devices.

Secondly, we provide two baseline algorithms in this paper. They are the approximate matching algorithm and the vertical shift approximate matching algorithm. Nonetheless, it is important to note that the two input sequences in our problem, i.e., the sensed and query sequences, have very different lengths while the baseline approximate matching and vertical shift approximate matching algorithms apply to time-series sequences of equal length, we integrate the sliding window design with these two algorithms to address our problem.

Also, to measure the performance of the inference algorithms, we take two types of errors into consideration. Specifically, (1). Type-1 Error, i.e., false negative, is corresponding to cases where the sensed sequence contains the event but the tested algorithm outputs false, i.e., fails to detect the event, for a given value of $\varepsilon$, (2). Type-2 Error, i.e., false positive, is corresponding to cases where the sensed sequence does not contain the event, yet the tested algorithm outs true, i.e., makes a wrong detection, for a given value of $\varepsilon$.

Last but not least, we discuss the various factors that may affect the performance of the inference algorithms. Note that these factors are important not only for understanding the degree of privacy leakage through sharing pedometer readings of wearable devices, but for designing defensive mechanisms as well.

1.2.2 Summary of Contributions

In this paper, we make the following contributions:

- We initiate the study of a novel problem on the privacy disclosure, specifically the
inference of sensitive human behavior events, from data sensed by wearable devices such as activity trackers.

- We provide two inference algorithms in this paper. They are the baseline approximate matching and the vertical shift approximate matching algorithm. Both are integrated the sliding window design to address our problem.

- We perform several experiments over real-world data that retrieved from one of the most popular devices on the market.

- Through real-world experiments, we find that the pedometer readings captured and shared by an off-the-shelf activity tracker indeed enables the accurate inference of events such as grocery shopping, walking to coffee shop, visiting gym, etc.

- We discuss the various factors that may affect the accuracy of inference algorithms and in turn can be leveraged for the design of defensive mechanisms.

1.3 Privacy Implications of Hotel Bookings

Problem Motivation: In this paper we investigate a novel problem on the implications of the information asymmetry model with transparency strategies, which has not been studied before. In this paper we focus on the OTA’s hotel booking business while our investigation indicates that top OTAs widely adopt the information asymmetry model in the hotel-booking business.

![Figure 1.2: Demonstrate of a hot deal from Hotwire.com](image-url)
Before introducing our technical results, we would like to demonstrate a simple attack on Hotwire.com, a very popular OTA in the United States. We choose Hotwire not only because it has very large market share, but also it provides a booking system applied asymmetry models with transparency strategies, which is called “Hotwire Hot Rates”. As Figure 3.2 shows, it allows users specify the destination and date, then it mostly returns a list of hotels with attracting discount price. From the perspective of users, it is acceptable that hidden hotel information leverages the price. For example, Figure 1.2 shows the screenshot returned for a search query that location = “Arlington, VA”, check-in date = “02/25/2019” and check-out date = “02/26/2019”. As shown in Figure 1.2, users could only make the decision by limited hotel information, e.g., hotel class, general location, discount price, key amenities (e.g. Free Internet, Airport shuttle, Business center, Fitness center, Golf nearby, Restaurant, Smoke-free rooms and accessibility for disabled) and etc. Figure 1.3 shows the hotel that users book. A 3-star hotel named “Holiday Inn National Airport/Crystal City” located in “Crystal City”. It has Free Internet, Airport shuttle, Business center, Fitness center, Golf nearby, Restaurant and Smoke-free rooms, along with accessibility for special needs. Furthermore, after a 45% discount from the original price, the discount price also matches the price shown in Figure 1.2. In this paper, we shall demonstrate how one can identify the hotel by using our algorithm. By using our algorithm, the booking system that targets high-end consumers would become less attractive since lower-cost are not equal to purchase unpredictable products any more.
1.3.1 Outline of Technical Results

Firstly, instead of assigning equal weight to hotel attributes, we assign proper weight to each hotel attribute. We demonstrate through real-world experiment that it achieves high success rate to infer the name of hotels before actual transaction, which raise a significant concerns on the current information asymmetry model.

Secondly, we propose two algorithms in this paper. They are the baseline inference algorithm and the IHF-matching algorithm. The IHF-matching algorithm assigns proper weight to each hotel attribute based on the frequency of attribute’s appearance in a specific region.

1.3.2 Summary of Contributions

In this paper, we make the following contributions:

- We introduce a formal definition of the information asymmetry model.

- Even the hotel identity is concealed under the information asymmetry model, and by inputting limited known hotel information, we successfully identified the name of hotel before the purchase.

- We perform several experiments over real-world data from one of the most popular OTAs on the market.

1.4 Privacy-preserving Framework

In this paper, we address privacy issues related to ranked retrieval model in web databases, each of which takes private attributes as part of input in the ranking function. Many web databases have both public and private attributes which serve different purposes. These websites, which are the owners of web databases, show the public attributes but keep private attributes invisible to the public. For example, social network websites provide privacy
settings which allow users to control the visibility of user profiles by hiding certain attribute values from public view. In order to maximize the protection effect, these websites also hide private attributes in query results so that the public can only access attributes that are set to public by users. Many websites believe that the adversary is unable to reveal the private attribute values from query results though private attributes have been taken as part of input in the ranking function. They declare that the private attributes are well protected. Intuitively, users indeed cannot view others’ private attribute values and their own private attribute values are hidden from public view. Users trust these websites because they believe that “what you see is what you get”, and are persuaded to input sensitive personal information as private attributes to databases. However, the investigation in [12] proved that though the values of private attributes could be hidden from public view, they still can be inferred from the ranked results.

1.4.1 Outline of Technical Results

In this paper, we propose a formal definition of adversaries. We divide adversaries into two categories: 1. The first category of adversaries are those who have no prior knowledge of attributes. Thus, adversaries cannot validate the authenticity of any tuple. We refer to this class of adversaries as \textit{domain-ignorant adversaries}. 2. The second category of adversaries are those who have prior knowledge of non-trivial attributes. This category of adversaries is able to validate the authenticity of an attribute value. We refer to this category of adversaries as \textit{domain-expert adversaries}.

Consider the \textit{domain-expert adversaries}, we study the ability of domain-expert adversaries and how this ability can break the privacy guarantee of the privacy-preserving framework with virtual tuples. Then we introduce a new implementation of privacy-preserving framework with true tuples.

We have carefully investigated the privacy guarantee of our framework. And we analyze its utility loss and propose an optimal solution to minimize the utility loss. We give a
heuristic algorithm of constructing the equivalent sets for this privacy-preserving framework

1.4.2 Summary of Contributions

In this paper, we make contributions as follow.

- We propose a formal definition of adversaries. We divide the adversaries into two categories based on their prior knowledge and investigate the ability of each adversary.

- We introduce a privacy-preserving framework. We propose a privacy guarantee and prove that our framework achieves this privacy guarantee.

- We introduce an implementation of the framework with true tuples. And we also give a heuristic algorithm of the framework with true tuples.

- We perform experiments over real-world. The results show a significant privacy-preserving improvement.
Chapter 2: Privacy Disclosure from Wearable Devices

2.1 Background

The popularity of wearable devices have seen an explosive growth in recent years, expanding from a somewhat “niche” market offering for athletes and patients (e.g., who have heart disease and need to constantly monitor their heart rates) to mass-market products that are estimated to become a 12.6 billion dollar business by 2018 [13]. Existing consumer-oriented wearable devices range from activity trackers (e.g., Fitbit, Withings, Jawbone Up) to full-fledged smart watches (e.g., Android Wear and Apple Watch) to even more futuristic offerings such as Google Glass. Wearable devices have significantly changed our daily lives [14, 15, 16].

Corresponding to such growth in popularity, more and more sensors and features are packed into wearable devices - e.g., pedometer, gyroscope, accelerometer, altimeter, compass, GPS, and heart rate monitor. These sensors work together to quietly monitor various aspects of the user’s everyday life, from the amount of sleep to the number of steps walked, from the number of floors climbed to the style of driving, etc. To offer users with convenient access and visual analytics of the sensed data, most wearable devices automatically synchronize the sensed data with a server maintained by the device manufacturer (through Internet connections available at mobile phones, computers, or standalone cellular connections featured in some wearable devices). Recently, there are even standardized APIs developed (e.g., Jawbone Up API [17]) that allow such sensed data to be conveniently shared with third-party applications such as RunKeeper, MyFitnessPal, etc. Furthermore, since social features form an integral part of many mobile applications, most wearable devices also allow users to share the sensed data (or an aggregate view of them) with friends. For example, activity trackers such as Fitbit and Up allow users to share with each other the number of steps walked, the amount of calories burned, etc., so as to motivate users to have a more
active lifestyle.

Just like the usefulness and “power” of wearable computing stems from its “closeness” to everyday life (e.g., being constant-on, monitoring vital physiological signs such as heart rate), such closeness may also raise serious privacy concerns. It should be noted that wearable device manufacturers already took into account a number of privacy concerns when designing their products. For example, almost all wearable platforms (e.g., Android Wear, Apple iOS) require explicit authorization from users before sharing sensitive information like location with others [18]. Some more sensitive parameters, e.g., heart rate (which can trigger serious implications if learned by external entities such as insurance companies), are simply excluded from the social sharing features (e.g., on Samsung S Health).

2.1.1 Time Series

A time-series data sequence is formed by a number of measurements taken over a given time interval. Let $X$ be a time-series sequence $X = \{x_1, x_2, ..., x_k\}$, where $x_i$ is a measurement taken at time $i$. We use $X[a : b]$ ($a < b$) to denote a subsequence of $X$ which starts from $x_a$ and ends with $x_b$, i.e., $X[a : b] = \{x_a, ..., x_b\}$. Examples of time-series data include stock prices,
process monitors, biomedical signals, and most importantly in this paper, the measured pedometer readings over a user’s daily life. An example of such measured pedometer readings is illustrated in Figure 2.1. The two most important characteristics of time-series data are continuity and (chronological) order between data points on the time domain.

2.1.2 Euclidean Distance

To properly determine whether an event of interest has occurred within a sequence of measured user activities, it is important to measure the distance (or similarity) between two time-series (sub-)sequences. The Euclidean distance is arguably the most common distance metric used in the literature. Specifically, consider two time-series sequences $X = \{x_1, x_2, ..., x_k\}$ and $Y = \{y_1, y_2, ..., y_k\}$, where $x_i$ and $y_i$ represent the value of $X$ and $Y$ at the same time slot $k$, respectively. The Euclidean distance between the two sequences, $D(X, Y)$, is defined as

$$D(X, Y) = \left( \sum_{i=1}^{k} |x_i - y_i|^2 \right)^{\frac{1}{2}} \quad (2.1)$$

2.2 Problem Definition

Our main focus in the paper is to examine whether a pre-defined event, e.g., walking from office to a nearby coffee shop, can be identified from amongst many other routine or random activities undertaken by the user and captured by the wearable device. Intuitively, there are a number of features captured by a pedometer that could potentially lead to the detection and the recognition of such a pre-defined event. Firstly, an individual’s step length is relatively stable. That means their average number of steps per minute is also stable if they walk about the same length. Secondly, many people walk from their offices to the nearby coffee shop in exactly the same route everyday. That means they probably stop at the same crossroad during their walk. As a result, a fluctuation will repeatedly appear in their historic activity data. Based on these features, anyone with access to the pedometer outputs, even at a coarse
granularity such as one minute (which is the granularity supported by many applications
APIs).

To formulate this problem, we consider two input time-series sequences. One is a long
time-series sequence which we refer to as the sensed sequence \( S = \{s_1, s_2, \ldots, s_N\} \) of length
\( N \). This sequence captures all activities undertaken by the user during a pre-defined period
of time (e.g., one day). Another input is a much shorter sequence, which we refer to as the
query sequence \( Q = \{q_1, q_2, \ldots, q_n\} \) of length \( n \). This sequence represents the (sensitive)
event that an adversary aims to infer from the user’s sensed activities. For example, \( Q \) may
be a sequence of pedometer readings when the user (or an adversary) walks from the user’s
office to the nearby coffee shop. The goal of the adversary is to infer from the two input
sequences a binary output on whether the query sequence appears in the sensed sequence -
i.e., 1 represents that the event is part of the monitored sequence (e.g., the user did walk to
the coffee shop during the day of interest), while 0 represents the opposite (resp., the user
did not travel there during the day).

2.3 Baseline Algorithms for Event Detection from Time-Series Data

We now review two baseline algorithms for detecting a pre-defined event from a user’s time-
series activity data. Because of the non-deterministic nature of pedometer outputs (which are
subject to the inherent, sometimes significant, error of consumer-grade pedometers [19, 9]),
it is important for the event detection algorithm to properly measure the similarity between
two approximate, rather than exactly the same, sequences [20]. In this paper, we consider
two baseline algorithms extensively studied in the literature for this approximate matching
problem: approximate matching [20] and vertical shift approximate matching [21].

2.3.1 Approximate Matching algorithm

The Approximate Matching algorithm simply considers a distance metric, e.g., the afore-
mentioned Euclidean distance \( D(X, Y) \), between two time-series sequences \( X \) and \( Y \), and
then determine them to be a match if and only if the distance falls below a pre-determined threshold $\varepsilon$ ($\varepsilon \geq 0$) \cite{22}. While simple, the approximate matching algorithm requires the amplitude of input sequences to be close to each other - which might not hold in practice. For example, depending on how tightly wrapped the wrist wrap is, a wrist-worn device (e.g., Fitbit) might return very different readings for the same user activity.

### 2.3.2 Vertical Shift Approximate Matching

To address this problem, the second, slightly more complex algorithm is **Vertical Shift Approximate Matching**. This algorithm maintains the similarity between two time-series sequences $X$ and $Y$ so long as they match with each other (approximately, of course) after shifting one “down” to the other. An example of this scenario is demonstrated in Figure 2.2. One can see from the figure that, if the simple approximate matching is used, sequences $X$ and $Y$ will be not considered close because of the large Euclidean distance $D(X,Y)$. Vertical shift approximate matching, on the other hand, can address this issue and correctly identify the closeness between $X$ and $Y$.

More formally, the distance measure adopted by vertical shift approximate matching \cite{22},

$$D(X,Y) = \left( \sum_{i=1}^{k} ((x_i - y_i) - (x_A - y_A))^2 \right)^{\frac{1}{2}} \leq \varepsilon \quad (2.2)$$
where \( x_A = \frac{1}{k} \sum_{i=1}^{k} x_i \) and \( y_A = \frac{1}{k} \sum_{i=1}^{k} y_i \)

2.4 Implementation of Inference Attacks

Since the two input sequences in our problem, i.e., the sensed and query sequences, have very different lengths (i.e., \( N \gg n \)) while the baseline approximate matching and vertical shift approximate matching algorithms apply to time-series sequences of equal length, we integrate the sliding window design with these two algorithms to address our problem. Specifically, the sliding window technique produces subsequences (i.e., a sliding window) of length \( n \) from the sensed sequence \( S \) - i.e., \( \langle s_1, \ldots, s_n \rangle, \langle s_2, \ldots, s_{n+1} \rangle, \ldots, \langle s_{N-n+1}, \ldots, s_N \rangle \). We represent these subsequences as \( S[1, n], S[2, n+1], \ldots, S[N-n+1, N] \), respectively.

Given each subsequence \( S[a, a+n-1] \) \( (a \in [1, N-n+1]) \) generated by the sliding window, we apply approximate matching and vertical shift approximate matching algorithms, respectively, over the subsequence and the query sequence \( Q \) to identify if a match exists. So long as one subsequence \( S[a, a+n-1] \) returns true (i.e., has its Euclidean distance with the query sequence \( Q \) below a pre-determined threshold \( \varepsilon \)), we output true for the problem, indicating the detection of event within the sensed sequence \( S \). Algorithm 1 and Algorithm 2 depict the implementation of the two algorithms, respectively.
Algorithm 1: Approximate Matching pseudocode

Input: $\epsilon, S, Q, \text{Length}_S, \text{Length}_Q$

Output: Bool $\text{Matching}$

1. Initialize $\text{countMatches} = 0$;
2. for $i = 1; i \leq (\text{Length}_S - \text{Length}_Q + 1)$ do
   3. SlidingWindow($S[i, i + \text{Length}_Q - 1]$);
   4. ApproximateMatching($D(S[i, i + \text{Length}_Q - 1], Q)$);
   5. if ($D(S[i, i + \text{Length}_Q - 1], Q)) \leq \epsilon$ then
      6. $\text{countMatches} + +$;
   end
3. end
4. if ($\text{countMatches} > 0$) then
   5. return $\text{Matching} = \text{true}$;
   else
   6. return $\text{Matching} = \text{false}$;
end

Algorithm 2: Vertical Shift Approximate Matching pseudocode

Input: $\epsilon, S, Q, \text{Length}_S, \text{Length}_Q$

Output: Bool $\text{Matching}$

1. Initialize $\text{countMatches} = 0$;
2. for $i = 1; i \leq (\text{Length}_S - \text{Length}_Q + 1)$ do
   3. SlidingWindow($S[i, i + \text{Length}_Q - 1]$);
   4. VerticalShiftApproximateMatching($D(S[i, i + \text{Length}_Q - 1], Q)$);
   5. if ($D(S[i, i + \text{Length}_Q - 1], Q)) \leq \epsilon$ then
      6. $\text{countMatches} + +$;
   end
3. end
4. if ($\text{countMatches} > 0$) then
   5. return $\text{Matching} = \text{true}$;
   else
   6. return $\text{Matching} = \text{false}$;
end
Chapter 3: Privacy Implications of Hotel Bookings

3.1 Background

In the last decades, the prosperity of information technology has significantly revolutionized the traditional tourism distribution [23, 24, 25, 26]. One of the most notable intermediaries, known as the Online Travel Agents (OTAs), specialize in selling tourism products and services online by embracing state-of-art techniques. In 2018, online travel sales generated 564.87 billion U.S. dollars in total. The U.S. alone contributed around 190.4 billion U.S. dollars in 2018 [27]. The OTAs gradually became an important part between the suppliers and end consumers. The growth of the online tourism industry has attracted lots of studies on business model [28, 29], tourism distribution channel [30, 31], customer booking behaviour [32, 33], etc.

In the tourism distribution industry, information asymmetry is the essence of revenue method for both traditional agents and OTAs. Travel agents possess more information than customers to create profits. Nowadays, most OTAs still make profits by using typical information asymmetry models in e-Commerce, for example, intentionally manipulating products’ price by concealing the knowledge of demand (e.g. the amount of actual transactions, the view of products, etc.) to stimulate sales. However, in recent years there have been many leading OTAs developing novel tourism products that produced by new asymmetry models with transparency strategies. Prior research has primarily focused on the influence of information asymmetry [34, 35] and the effectiveness of information asymmetry [36, 37, 38] in tourism distribution industry.

In this paper we investigate a novel problem on the implications of the information asymmetry model with transparency strategies, which has not been studied before. In this paper we focus on the OTA’s hotel booking business while our investigation indicates that top OTAs widely adopt the information asymmetry model in the hotel-booking business.
We introduce a formal definition of the information asymmetry model, and we show how information can be compromised by the inference attack in a seemingly well-defined design of such kind of information asymmetry models.

To understand how the privacy leaks in the asymmetry model, we would like to briefly introduce the OTAs’ hotel booking system. The system prompts the user to input searching criteria, then return a list of hotels based on user inputs. Note that databases in the booking system have both public and private attributes, which categorized by the transparency strategies the OTAs apply. Specifically, the transparency strategies are to reveal or conceal product attributes (e.g. price, geographic information, amenities, customer feedback, etc.) to the user. On the one hand, for OTAs, new products with controlled information but lower price may boost sales. On the other hand, for customers, products with incomplete information are also acceptable because they pay less for a stay although it comes with unpredictability.

In the real world, flexible transparency strategies are adopted by individual OTA based on its business philosophy and market position, as well as target different kinds of consumer needs. For example, as shown in the Figure 3.1, Priceline (Priceline.com) allows consumers to bid with limited information about the price, the area and the property class. The detailed hotel information will be presented to customers once the system accept the bid.

Another example, as shown in the Figure 3.2, is the booking interface of Hotwire (hotwire.com), which conceals the name, the precise location and partial amenities of the property until a customer completes the transaction. Correspondingly, Hotwire offers a discount price to compensate the concealed information. Apparently, both Hotwire and Priceline adopt the same level of product transparency and supplier transparency, but Hotwire adopts higher price transparency than Priceline.

The problem here, however, is that many OTAs release detailed hotel information to partners, meanwhile they provide a hotel booking system which lists detailed hotel information on their own websites for consumers who are willing to pay a premium price for predictable
The purpose of launching this kind of booking system is, understandably, to maximize profits by satisfying high-end consumers’ needs. From the perspective of privacy, this kind of model is harmless. After all, even the information asymmetry model might take detailed information as input, with certain level of transparency, and its output to customers is a list of hotels which can not be directly identified. Naturally, the OTAs believe that the certain information asymmetry is guaranteed.

In our investigation of real-world online hotel booking systems, we found that this
belief is incorrect. In this paper, even the hotel identity is concealed under the information asymmetry model, and by inputting limited known hotel information, we may successfully identify the name of hotel before the purchase.

3.2 Preliminaries

In this section, we start with introducing the formal definition of the asymmetry model with transparency strategies. Then we briefly review the matching algorithm that has been proposed in the literature.

3.2.1 Asymmetry model with transparency strategies

As we discussed in the introduction, the databases store two kinds of attributes: some attributes that OTAs publicly use to demonstrate basic information about the hotel (e.g. a few amenities, approximate location, property class, etc.), and some attributes that remain private (e.g. hotel name, some hotel detailed description, etc.). Thus, both public and private attributes are stored in the database. Note that public and private features may vary by different transparency strategies for a single attribute. For example, in the Figure 3.1, the price is a private attribute in the database, however, in the Figure 3.2, the price becomes a public attribute.

Consider an \(n\)-tuple (i.e., \(n\)-hotel) database \(D\) with a total of \(m + m'\) attributes, including \(m\) public attributes, in which \(A_1, \ldots, A_m\) attributes are public, and \(m'\) private attributes \(B_1, \ldots, B_{m'}\) which are concealed to the user, by applying the transparency strategy \(T_i\). We use \(h[A_i]\) (resp. \(h[B_i]\)) to denote the value of attribute \(A_i\) (resp. \(B_i\)) for a hotel \(h(h \in D)\). Note that there is no duplicate hotel in the database (we can achieve this by assigning a unique hotel \(id\) for each hotel in the real-world database). For the purpose of this paper, we assume that \(D\) does not change during the inference attack.

As we introduced in the introduction, the user is prompted to input some basic search criteria (i.e. city, state, check-in date, check-out date, etc.). The database returns a list of
hotels based on user inputs. We formalize the model as follows, given a supported query $q$, the database computes the searching function $s_{T_i}(h|q)$ for each hotel $h$ ($h \in D$) by applying the transparency strategy $T_i$, then returns a list of hotels (i.e. some of which might not exactly match the query requests). Of course, for each hotel in the result list, not only the hotel name is concealed, but also only the $m$ public attributes are displayed. For the purpose of this paper, we assume that the searching function $s_{T_i}(h|q)$ is well defined. There are many studies related to this kind of searching function, for example, [39]. It is important to point out that our inference attack is working on arbitrary searching functions (e.g. approximate rather than exactly the same result, ranked/unranked result, etc.) if the searching results include public attributes.

3.2.2 Matching Algorithm

We now review a matching algorithm for finding the hotel from a returned list of hotels. As we demonstrated in the introduction, OTAs may release the completed hotel dataset to partners, or sometimes even provide detailed hotel information on their own websites for high-end customers. Therefore, it is possible to retrieve the dataset by APIs and web crawlers. Because only public attributes are revealed in the returned interface, it is important for the hotel matching algorithm to properly find the possible target hotel between the returned list and the hotel dataset. Specifically, consider $h[A_i]$, which is the public attribute value of a hotel $h$ in the dataset, and $r[A_i]$ which is the public attribute value of the target hotel $r$ in the returned list. The matching score between $h$ and $r$ is defined as:

$$\text{Score}(r|h) = \sum_{i=1}^{m} \alpha(r[A_i], h[A_i])$$  \hspace{1cm} (3.1)

The algorithm simply considers a distance measure for each public attribute, i.e., the $\alpha(r[A_i], h[A_i])$, is a variation of discrete metric, (1) $\alpha(r[A_i], h[A_i]) = 0$ if $r[A_i] \neq h[A_i]$, and (2) $\alpha(r[A_i], h[A_i]) = 1$ if $r[A_i] = h[A_i]$.

We would like to note again that the adversary has no knowledge of the searching

21
function design. The inference attack is based on the returned list and the hotel dataset.

### 3.3 Problem Definition

Our main purpose in the paper is to find the detailed information of the target hotel, which is shown in the returned list and only public attributes are displayed to users. Specifically, a number of hotel information is hidden because of applying the asymmetry model with transparency strategy, e.g., hotel name, hotel address, full introduction of hotel, reviews from guests, etc. Intuitively, we could possibly retrieve the detailed information by inferring hotel name from a known hotel dataset. Firstly, the value of a hotel’s attributes are relatively stable. Due to the consideration of keeping normal operation and budget, the owner of the hotel seldom renovates and upgrades the property. Secondly, in the real-world, OTAs prefer to categorize some of the popular amenities that users are always looking for as public attributes, e.g., access to the Internet, free breakfast included, fitness center, etc. That means even if the owner of the hotel did major renovations, the public attributes are unchanged. Moreover, OTAs update the database frequently, because inaccurate hotel information would diminish the attraction and revenue, and in turn could have an impact on future business. Based on these reasons, along with our assumption in the preliminaries that the database does not change during the inference attack, the adversary can infer the target hotel by our algorithm.

**Definition 1.** Given a database $D$ and a target hotel $r (r \in D)$, where $r[A_1], \cdots, r[A_m]$ are public known, find a hotel $h (h \in D)$, such that $r[A_1] = h[A_1], \cdots, r[A_m] = h[A_m]$, where $V^A_i$ and $V^B_j$ are the attribute domain for $A_i$ and $B_j$, $V^A_i$ and $V^B_j$ are discrete and publicly known.
3.4 Baseline Inference and IHF-matching Algorithm

3.4.1 Baseline Inference Algorithm

In the Section 3.2.2, we introduced the matching algorithm. The Baseline Inference Algorithm is based on the matching algorithm. The Baseline Inference Algorithm simply consider a distance metric, e.g., the aforementioned Matching Algorithm $Score(r|h)$, between two hotels $r$ (target hotel) and $h$ (known hotel in the dataset), and then determine them to be a match if and only if the $Score(r|h) = 0$.

3.4.2 IHF-matching algorithm

In the baseline algorithm, we simply consider all attributes are equally important. However it is known that certain attributes that satisfy customer basic needs or have become standard infrastructures (e.g., access to the Internet, parking, etc.) may appear a lot in the description of hotels. In this case, two distinct hotels may have the same score if they have the same number of public attributes displayed in the hotel description. Therefore, the challenge is to distinguish two distinct hotels from the returned list even if they have the same number of public attributes. To address this challenge, we introduce the IHF-matching algorithm in this paper. Once again, we would like to note that the domain of public attributes is Boolean type. We leave the discussion of numeric public attributes to the future work.

IHF stands for Inverse Hotel Frequency, and the IHF-matching algorithm is based on computing IHF weight that used in matching functions. This weight is a statistical measurement used to evaluate how important an attribute is to a group of hotels (e.g. a group of hotels located in the same region). The importance diminishes proportionally to the frequency of an attribute that appeared in the group of hotels.

Algorithm 3 and Algorithm 2 depict the implementation of the two algorithms, respectively.
**Algorithm 3: Matching Algorithm pseudocode**

**Input:** $D, r$

**Output:** $HotelList$

1. Initialize $MaxScore = 0$, $HotelIdList$;
2. for $i = 1; i \leq n$ do
3. 
4. 
5. 
6. 
7. 
8. 
9. 
10. 
11. 
12. 
13. 
14. 
15. 
16. 
17. 
18. 
19. 

**Algorithm 4: IHF-matching Algorithm pseudocode**

**Input:** $D, r, map$

**Output:** $HotelList$

1. Initialize $MaxScore = 0$, $HotelIdList$, $HotelList$;
2. for $i = 1; i \leq n$ do
3. 
4. 
5. 
6. 
7. 
8. 
9. 
10. 
11. 
12. 
13. 
14. 
15. 
16. 
17. 
18. 
19. 

24
4.1 Motivation

In this paper, we address privacy issues related to ranked retrieval model in web databases, each of which takes private attributes as part of input in the ranking function. Many web databases have both public and private attributes which serve different purposes. Websites, which are the owners of web databases, show the public attributes but keep private attributes invisible to the public. For example, social network websites provide privacy settings which allow users to control the visibility of user profiles by hiding certain attribute values from public view. In order to maximize the protection effect, these websites also hide private attributes in query results so that the public can only access attributes that are set to public by users. Many websites believe that the adversary is unable to reveal the private attribute values from query results though private attributes have been taken as part of input in the ranking function. They declare that the private attributes are well protected. Intuitively, users indeed cannot view others’ private attribute values and their own private attribute values are hidden from public view. Users trust these websites because they believe that “what you see is what you get”, and are persuaded to input sensitive personal information as private attributes to databases. However, the investigation in [12] proved that though the values of private attributes could be hidden from public view, they still can be inferred from the ranked results.

Motivation

According to the study by Rahman et al., one can infer the values of private attributes of a victim tuple by issuing well-designed queries through a top-k query interface. Rahman et al. discovered that under the premise that the ranking function satisfies both monotonicity condition and additively condition[12], the problem of excluding a value $\theta$ of a private
attribute $B_1$ in its domain can be reduced to finding a pair of differential queries $q_\theta$ and $q'_\theta$ which satisfy following properties: (1) they hold the same predicate on all attributes but $B_1$, (2) $q_\theta[B_1] = \theta$ while $q'_\theta[B_1] \neq \theta$, and (3) $q_\theta$ returns the victim tuple $v$ while $q'_\theta$ does not. An adversary therefore is able to infer the value of a private attribute by excluding all but one value in the domain if the domain is discrete and finite. Most research done to date has focused on the development and evaluation of effective ranking functions of the Ranked Retrieval model. Thus the discovery of this unprecedented privacy risk attracted our interest because it had not been adequately addressed by any existing defense techniques so far.

Before introducing our technical results, we would like to first review existing privacy preserving techniques. The most straightforward method for privacy preserving is completely removing all private attributes from ranking functions. In this case, the query results do not contain any information of private attributes. However, in practical, the benefit of adding such private attributes as inputs is obvious: higher effectiveness (e.g. dating sites may take sensitive private attribute race and religion into consideration during matching their users). Therefore, in this paper, we focus on preserving private attributes as well as maintaining the utility of ranking functions.

There has been extensive work on privacy-preserving in databases using data perturbation in which tuples in the databases are modified to protect sensitive information. Multiplicative Noise \cite{40} masks continuous data by adding variance to the original data. Micro-aggregation \cite{41} groups individual records into small aggregations and replaces the values of each record with the average value of each group. However, variance or averages cannot be computed for categorical data, especially non-numeric data. Therefore, Multiplicative Noise and Micro-aggregation cannot solve the problem where datasets contain non-numeric categorical data. Rule hiding \cite{42} proposed a strategy that reduces the confidence of rules that specify how significant they are, given the inference rules known. Categorical data perturbation \cite{43} proposed a guarantee of privacy by limiting the posterior probability of perturbed dataset, which suffers from high utility loss. Data swapping \cite{44} swaps the values of sensitive
records with non-sensitive records in order to protect privacy of the former while maintain summary statistic of the dataset. However, in our setting, our goal is to preserve privacy of private attributes of all records. Data shuffling [45] preserve privacy of numeric attributes by swapping their values with each other. This method, however, is limited by the distribution of datasets and subsequent utility loss.

Condensation is another approach of privacy-preservation data mining. [41] treats tuples as data points in a multidimensional space and tries to cluster all data points in a database. However, this method requires a distance function for tuples in a database. Therefore, attributes have to be numerical. Furthermore, we need prior knowledge to scale and standardize different attributes on different axes.

Suppression and generalization on databases have a number of studies [46, 47, 48]. This paper [46] analyzed the risk of releasing data without the consideration of re-identification by linking. [46] provided the $k$-anonymity protection model and [49, 50, 51] gave real-world systems which adopted the $k$-anonymity protection. Even the following research of [47, 48] proposed $l$-diversity and t-closeness, we can not ignore the fact that in a ranked retrieval model, namely, we take into account the private attribute values but we do not return the entire dataset regarding to a single query. Nevertheless, it is possible to directly apply the suppression and generalization on the dataset, but none of the approaches considered the optimization on ranked retrieval models.

Another widely studied approach in the academic area is the differential privacy [52]. It protected the privacy of individual when releasing statistical information of databases by adding noise. [53] then studied the algorithmic foundations of differential privacy. Moreover, [54] introduced a statistical framework for differential privacy. However, in practice, the differential privacy sometimes is not the first choice for the data owner because of its recondite.

An alternate approach is query auditing [55]. Query auditing is the process of examining past actions to check whether they were in conformance with official policies. In a specific
online database system, query auditing is the process of examining a user’s queries answered in the past and deny queries from the same user that could potentially cause a breach of privacy. However, it is inadequate to assume that an adversary is limited to only one account. In fact, many online databases impose no or loose restrictions on the number of accounts a user can create and the number of queries an account can issue every day. A recent study [12] shows that privacy can be compromised without breaking query auditing. As mentioned by [12], an adversary is able to infer private attribute values of any user in eHarmony[^1] with freely created accounts. In fact, the adversary conducts only two kinds of operations: creating accounts and issuing a number of queries, whose accesses are open to the public.

The limitation of any defense techniques for specific attacking methods (e.g. [56]) is that the defense technique eventually loses its effectiveness if the adversary changes the way of attack. In this paper, we remove the constraints on the methods of attack adopted by the adversary. In our framework, we allow a complete adversary that is able to compromise private attribute values through the top-$k$ results in arbitrary attacking methods. We also allow an harmless user do the same querying operations to acquire top-$k$ results as usual. Since attacking techniques can no longer be used to differentiate adversaries, we categorize adversaries by their prior knowledge - the domain of private attributes they are going to infer.

4.2 Privacy-preserving Framework

4.2.1 Ranked Retrieval Model

As discussed in the introduction, many web databases store both public and private attributes of users. In recent years, a large number of databases have been adopting the ranked retrieval model. Within proper ranking function, the system returns tuples that best satisfy the query (e.g. returns top-$k$ tuples). Consider an $n$-tuple (i.e., $n$-user) database $D$ with a total of $m + m'$.

[^1]: eHarmony is an online dating website. It was launched on August 22, 2000, and is based in Los Angeles, California.
attributes, including \( m \) public attributes \( A_1, \cdots, A_m \) and \( m' \) private attributes \( B_1, \cdots, B_{m'} \). Let \( V_i^A \) and \( V_j^B \) be the attribute domain (i.e., set of all attribute values) for \( A_i \) and \( B_j \) respectively. We use \( t[A_i] \) (resp. \( t[B_j] \)) to denote the value of a tuple \( t \in D \) on attributes \( A_i \) (resp. \( B_j \)). For the purpose of this paper, we assume there is no duplicate tuple in the database.

Our ranked retrieval model is formalized as follows. Given a top-\( k \) query \( q \), the model is able to compute a score \( s(t|q) \) based on a predetermined ranking function for each tuple \( t \in D \), and returns the \( k \) tuples with the highest \( s(t|q) \). In this paper, we consider a linear ranking function. Linear ranking function can be defined as:

\[
s(t|q) = \sum_{i=1}^{m} w_i^A \cdot \rho(q[A_i], t[A_i]) + \sum_{j=1}^{m'} w_j^B \cdot \rho(q[B_j], t[B_j]),
\]

where \( w_i^A, w_j^B \in (0, 1] \) are the ranking weights for attributes \( A_i \) and \( B_j \) respectively. In this paper, we consider the case that attributes \( A_i \) and \( B_i \) are categorical, and \( \rho(q[A_i], t[A_i]) = 1 \) if \( q[A_i] = t[A_i] \) (\( q[B_j] = t[B_j] \) respectively), or 0 if \( q[A_i] \neq t[A_i] \) (\( q[B_j] \neq t[B_j] \) respectively). \( \rho \) can be easily extended to numerical attribute cases in which \( \rho(q[A_i], t[A_i]) = \frac{1}{|q[A_i] - t[A_i]|} \) (\( \frac{1}{|q[B_j] - t[B_j]|} \) respectively). We note again that the ranking function follows the monotonicity and additivity properties.

### 4.2.2 Adversary Model

We mentioned that the adversary wants to compromise private attribute values of a victim tuple \( v \) through arbitrary attacks. Without loss of generality, we make the assumption that an adversary has knowledge about the ranking function and all the public attributes. Furthermore, we assume that the adversary is able to issue queries and insert tuples with specified values to the database.

As we discussed in section 4.1, prior knowledge of attributes will definitely help the adversary to perform a more effective attack.

For example, when \( \theta(V_j^B) \) is an uniform distribution, the prior knowledge of adversaries...
can be ignored. When $\theta(V_B^j)$ is a non-trivial prior distribution for $V_B^j$, the prior knowledge of adversaries possess can potentially be used to launch more effective attacks, e.g., for a victim tuple $t$, adversaries prefer to choose some $t[B_j]$ based on the distribution.

Thus we partition the adversary into two categories: (1) The adversary has no prior knowledge of attributes, (2) The adversary has prior knowledge of attributes.

We assume that the objective of the adversary is to maximize the following $g_A$ value

$$g_A = Pr(v[B_j] = a|\theta(V_B^j))$$

where $\theta(V_B^j)$ stands for the adversary’s prior knowledge of $V_B^j$. The prior knowledge may have diverse forms, e.g., correlations between $v[A_i]$ and $v[B_j]$. In this case, the adversary can infer $v[B_j]$ by knowing $v[A_i]$, where $v[A_i]$ is known in public. Here we assume that the adversary is able to validate the authenticity of $v[B_j]$ based on $\theta(V_B^j)$. This model can describe various kinds of adversaries because the adversary can possess the $|V_B^j|$, and thus can assume arbitrary value in $|V_B^j|$ be the true $v[B_j]$. The adversary can validate this arbitrary value by $\theta(V_B^j)$.

**Definition 2.** An adversary is domain-ignorant if the adversary has no prior knowledge of attributes. An adversary is domain-expert if the adversary has prior knowledge of non-trivial attributes.

**4.2.3 Problem Statement**

Ideally, we want to keep adversaries from retrieving the private attributes’ values of victim tuple $v$ from ranking results. Ideally, we want to keep adversaries from retrieving the private attributes’ values of victim tuple $v$ from ranking results. In practice, adversaries may retrieve the possible private attribute’ values through arbitrary attacks. However, if adversaries cannot 100% determine the values of $v$’s private attributes, we can conclude that privacy of private attributes is well preserved. We define the privacy of $v[B_j]$ as $P_{v[B_j]}$, which is
the possibility that an arbitrary adversary fails to infer the value of $v[B_j]$ \cite{57}. Therefore, $P_{v[B_j]} = 0$ represents the worst case where privacy of $v[B_j]$ is compromised, while $P_{v[B_j]} = 1$ represents an ideal case where an adversary is unable to infer the authentic value of $v[B_j]$.

In this paper, the objective of privacy preserving is to protect all private attributes of any tuple $t$. Therefore, we define the privacy-preserving problem as:

$$\forall t \in D, j \in \{1, \ldots, m'\}, P_{v[B_j]} \geq \varepsilon,$$  \hspace{1cm} (4.3)

where $\varepsilon$ is a constant that we present as a privacy guarantee.

### 4.2.4 Privacy-preserving Framework

In the framework, we want to keep the adversary from identifying a victim tuple $v$ from ranked results. We find that it can be achieved by finding at least another tuple $t$ (where $t \neq v$) which has the same ranking score as $v$ for all queries. In the query results, the rank of $v$ from the rank of $t$ for an arbitrary $q \in Q$, where $Q$ is the set of all possible queries:

$$\forall q \in Q, \text{Rank}(v|q) = \text{Rank}(t|q)$$

We now prove that $v$ and $t$ are indistinguishable if $v$ and $t$ are equivalent. Given two tuples $v$ and $t$, where $v[A_i] = t[A_i]$, and two databases $D_1$ and $D_2$, where $D_2 = (D_1 - \{v\} \cup \{t\})$.

Consider $v$ and $t$ are equivalent. By simply issuing queries and examining the ranked results, the adversary cannot distinguish $D_1$ from $D_2$. Therefore, $v$ and $t$ are indistinguishable.

As such, we introduce the construction of an Equivalent Set, which is to put the victim tuple $t$ into a set in which all tuples are equivalent in any ranked results. In this paper, we name this set as the Equivalent Set and denote it as $E_t$. We define Equivalent Set as follow:

**Definition 3.** *In a set $E_v = \{v, t_1, \ldots, t_k\}$, where $v[A_i] = t_1[A_i] = \cdots = t_k[A_i]$, and $v, t_1, \ldots, t_k$ are equivalent, we call the set $E_v$ as the Equivalent Set, and the domain of $B_j$ as $C^B_j$.*
To achieve privacy of \( v[B_j] \), since any technique based on ranked results cannot distinguish \( v, t_1, \ldots, t_k \) in \( E_v \), the privacy guarantee of \( v[B_j] \) is:

\[
P_{v[B_j]} = (1 - \frac{1}{|C_j^B|}) \times 100\% \quad (4.4)
\]

To achieve our guarantee of privacy preservation defined in (4.3), we have to make sure that a) for each \( t \in D \), \( t \) is included by one and only one equivalent set \( E_t \) and b) the privacy guarantee defined in (4.4) is valid for any \( j \in \{1, \ldots, m'\} \). Note that for the condition a, if \( t \) is not included by any \( E_t \), then obviously \( t \) is not protected. Our privacy guarantee cannot be achieved. Also, if \( t \) appears in more than one \( E_t \), e.g., \( t \in E_t \) and \( t \in E'_t \), then according to the definition[3] all elements in \( E_t \) and \( E'_t \) are equivalent and, thus, \( E_t \) and \( E'_t \) should be merged into a new \( E''_t \).

In the privacy-preserving framework, except the victim tuple, the other tuples in the equivalent set could be either virtual tuples or true tuples. We will give details of the implementations which construct equivalent set with true tuples.

### 4.2.5 Utility Loss Measurement

To quantify utility loss provided by a method, we use a measurement based on the modification of ranked results. Given a database \( D \) and a set of all possible queries \( Q \), we compare the ranked results of \( Q \) over \( D \) before and after applying a privacy-preserving method. We define utility loss as follow:

\[
U = \sum_{t \in D} \sum_{q \in Q} |\text{Rank}(t|q) - \text{Rank}'(t|q)|
\quad (4.5)
\]

where \( \text{Rank}(t|q) \), \( \text{Rank}'(t|q) \) refer to the ranks of tuple \( t \) given query \( q \) before and after applying our privacy-preserving frameworks respectively.
4.2.6 Framework with virtual tuples

In the paper [57, 58], we introduced a framework with virtual tuples.

An intuitive algorithm to generate a equivalent set of size \( n + 1 \) for tuple \( v \) is to generate virtual tuples \( t_1, \ldots, t_n \) such that \( t_i[B_j] \neq v[B_j], \forall i \in \{1 \ldots n\} \) and \( j \in \{1 \ldots m'\} \). Specifically, let the initial \( E_t = \{v\} \). Then we can generate a virtual tuple \( t_1 \) by assigning each \( t_1[A_i] \) with \( v[A_i] \) and each \( t_1[B_j] \) with a value randomly picked from \( V_j^B \setminus \{v[B_j]\} \) for all \( i \in \{1, \ldots, m\} \) and \( j \in \{1, \ldots, m'\} \). In order to achieve privacy guarantee \( P_{v[B_j]} \geq \varepsilon \), we can further generate more virtual tuples \( t_2, \ldots, t_n \) to enlarge each \( C_j^B \) until \( 1 - \frac{1}{|C_j^B|} \geq \varepsilon \) for \( j \in \{1, \ldots, m'\} \). We name \( \text{minimum}\{|C_j^B|\}, \forall j \in \{1, \ldots, m'\} \) as the cardinality of \( E_t \).

In this paper [57], we prove its privacy guarantee \( P_{v[B_j]} \geq 1 - \frac{1}{l} \) for \( \forall t \in D \) and \( \forall j \in \{1, \ldots, m'\} \). According to (4.3), a privacy level of \( 1 - \frac{1}{l} \) can be achieved.

4.3 Privacy-preserving Framework with True Tuples

The feasibility of privacy-preserving framework is established in Section 4. Recall that the domain-expert adversaries hold the prior knowledge of non-trivial attributes. In Section 4.3.1, we firstly study the ability of domain-expert adversaries and how this ability can break the privacy guarantee of the privacy-preserving framework proposed in Section 4. In Section 4.3.2, we introduce a new implementation of privacy-preserving framework and prove its privacy guarantee. In Section 4.3.3, we analyze its utility loss and propose an optimal solution to minimize the utility loss. We give a practical heuristic algorithm of constructing the equivalent sets for this privacy-preserving framework in Section 4.3.3.

4.3.1 Domain-expert adversaries

As we mentioned in Section 4.2.2, domain-expert adversaries can view public attributes, as well as they are able to validate the authenticity of an attribute value. Here, we start by showing that if adversaries hold the prior knowledge of the non-trivial attributes correlation,
the privacy guarantee $P_{v[B_j]}$ for the equivalent set with virtual tuples cannot be achieved.

We consider a simple case that domain-expert adversaries can validate a private attribute $B_1$ by knowing a set of public attributes $S_A$. We denote the prior knowledge of the attributes correlation as $S_A \Rightarrow B_1$, where $S_A$ is a determinant set of public attributes, and $B_1$ is a dependent private attribute. Note that this attributes correlation exists when there are functional dependency and/or inevitable dependency between attributes in the dataset. In reality, the adversary can acquire such prior knowledge of non-trivial attributes correlation by being/consulting a domain expert, or using data mining techniques [59] to explore correlations between attributes. For example, based on the personal information (e.g. gender, ethnicity, age, blood type which can decide the gene) stored as public attributes and published in public medical data repositories, genetic epidemiologists can generally conclude that a dominant gene that is associated with a particular disease by the candidate-gene approach. Therefore, domain-expert adversaries could know that some candidate values violate the correlation of $S_A \Rightarrow B_1$.

As stated in Section 4.2.6, we can construct an equivalent set $E_v = \{v, t_1, \ldots, t_k\}$, where $v \in D$, $t_1, \ldots, t_k \not\in D$ and $v[B_j], t_1[B_j], \ldots, t_k[B_j] \in V_1^B$. We assume that domain-expert adversaries can retrieve $C_{B_j} = \{v[B_j], t_1[B_j], \ldots, t_k[B_j]\}$ by arbitrary attacking methods. From discussed above, according to $S_A \Rightarrow B_j$, domain-expert adversaries could know that:

$$\exists v[B_1] = C_{B_1}^B \setminus t_e[B_1], e \in \{1, \ldots, k\}$$ (4.6)

We now can prove that the ESVT loses its privacy guarantee $P_{v[B_1]}$ when facing the attack from domain-expert adversaries:

$$P'_{v[B_1]} = (1 - \frac{1}{|C_{B_1}^B| - k}) \times 100 < P_{v[B_1]}, \text{ where } 1 \leq k < |C_{B_1}^B|$$ (4.7)

Remember that when constructing ESVT, we generate equivalent tuples for each $v$ merely based on query workload and the assumption of domain-ignorant adversaries. As
shown above, once $t_e[B_1]$ can be excluded from $C^B_1$ when known $S_A \Rightarrow B_1$, the privacy guarantee loses.

Especially, it is a considerable detrimental threat to the ESVT of cardinality 2. Under this circumstances, suppose domain-expert adversaries can retrieve $C^B_j = \{v[B_j], t[B_j]\}$, one can notice that there are only two candidate values in $C^B_j$: $v[B_j]$ are targets, and $t[B_j]$ are virtual values. Assume that domain-expert adversaries hold the prior knowledge of correlation $S_A \Rightarrow B_j$, in the worst-case scenario, domain-expert adversaries know that every $t[B_j]$ is invalid. Therefore $\forall v[B_j] = C^B_j \setminus t[B_j]$, domain-expert adversaries then can conclude that $C^B_j = \{v[B_j]\}$. Recall that the adversary’s goal is to maximize the Function (4.2). In this case, adversaries would have $g_A = 100\%$ for every $v[B_j]$.

So far, we have seen how domain-expert adversaries could break the privacy guarantee mentioned in (4.4), and thereby cause the detrimental consequence of privacy disclosure. In reality, there are a number of prior knowledge that potentially can be used to validate the private attribute. In following sections, to protect privacy under attacks from domain-expert adversaries, we propose and give the design of constructing equivalent set with true tuples. Then we prove that the privacy guarantee is achieved no matter what prior knowledge domain-expert adversaries hold. We also give an optimization algorithm and investigate the cost.

4.3.2 Privacy-preserving Framework Design

We have shown above that domain-expert adversaries can break the privacy guarantee of ESVT in Section 4.2.6 so that the necessity of designing a robust privacy-preserving framework is unquestionable. In this subsection, we propose a framework that constructs equivalent sets with true tuples, which we will refer to as Equivalent Set with True Tuples (ESTT). This framework offers the same degree of privacy guarantee for private attributes as mentioned in (4.4), even under the attack from domain-expert adversaries. We must point out that the new design is different with the design of ESVT in Section 4.2.6. Intuitively,
each equivalent set consists of true tuples only in this new framework. In other words, here we avoid generating tuples that violate the prior knowledge of domain-expert adversaries. Instead, we use existing tuples in the dataset because they comply with potential prior knowledge of domain-expert adversaries (e.g. the attributes correlation as we discussed in Section 4.3.1). Note that in this paper, we only consider that users are willing to input true information which complies with prior knowledge.

**Definition 4.** If there are at least \( l \) “valid” candidate values for every private attribute in equivalent sets, we call it \( l \)-candidate equivalent set, where \( l \leq \min(|V_j^B|), j \in \{1, \ldots, m'\} \).

As we mentioned in Section 4.2.1 in the ranking model, for each tuple \( t, t[B_1], \ldots, t[B_j] \in V_j^B \). And in ESTT, \( t \in D \). Therefore, one can find out that the maximal \( |C_j^B| \) is the minimum \( |V_j^B| \). Therefore, \( l \leq \min(|V_j^B|) \) in the \( l \)-candidate equivalent set.

**Privacy Guarantee:** The \( l \)-candidate equivalent set can achieve the same privacy guarantee as the framework with virtual tuples even adversaries are domain-expert adversaries. We denote the equivalent set as \( E_v = \{v_1, \ldots, v_k\} \), where \( v_1, \ldots, v_k \in D \). Note that, the same as constructing \( E_v \) in Section 4.2.6 in each \( E_v \) we make \( s(v_1|q) = s(v_2|q) = \cdots = s(v_k|q) \). It is exceedingly important that \( v_1, \ldots, v_k \) share the same public attributes. One can find out that all values in \( C_j^B \) satisfy \( S_A \Rightarrow B_j \) because \( v_1, \ldots, v_k \) are true tuples. Adversaries have to conclude that \( v_1[B_j], \ldots, v_k[B_j] \) are possible true values for \( v_1 \). Therefore, \( S_A \Rightarrow B_j \) can not be used to exclude any candidate values from \( C_j^B \). Here, we give our privacy guarantee. Assume that the target of adversaries is \( B_j \), and adversaries are able to retrieve \( C_j^B \) by arbitrary attacking methods. In this case, since there are at least \( l \) “valid” candidate values for private attribute \( B_j \), which means \( |C_j^B| \geq l \), according to (4.4):

\[
P'_{v_1[B_j]} = \left( 1 - \frac{1}{|C_j^B|} \right) \times 100\% \geq \left( 1 - \frac{1}{l} \right) \times 100\%
\]

(4.8)

The privacy guarantee is achieved by ESTT. As we mentioned above, the framework can defend attacks from domain-expert adversaries who hold the prior knowledge of attributes.
correlation. We now extend the general assumption that domain-expert adversaries can hold any non-trivial prior knowledge. Consider the simplest ESVT, which is

2-candidate equivalent set. It guarantee that at least 2 valid candidate values for each $B_j$. Thus, according to (4.8), $P'_{v_1 | B_j} \geq (1 - \frac{1}{2}) \times 100\%$. The privacy guarantee is achieved. Furthermore, for adversaries, the goal is to maximize the Function (4.2). In this case, adversaries would have $g_A = 50\%$ for every $v_1 | B_j$. Obviously, we limit the probability of adversaries to get the correct private attribute value of the victim tuple to 50%.

Secondly, it is important to set proper size for l-candidate equivalent set when considering the practicability in real web databases. Admittedly, if we put $n'$ tuples that share the same public attributes into a single equivalent set $E_v$, it could satisfy the l-candidate requirement. However, it is impractical because every tuple in $E_v$ will return the same score since $s(v_1 | q) = s(v_2 | q) = \cdots = s(v_{n'} | q)$. Thereby the ranking function loses its functionality. Here we introduce the set size $k$, which is the number of tuples in a l-candidate equivalent set.

**Definition 5.** When a l-candidate equivalent set contains $k$ different true tuples respectively, we call it l-candidate equivalent set with $k$-tuples, where $k \geq l$, and $l \leq \min(|V^B_j|), j \in \{1, \ldots, m'\}$.

Note that each $E_v$ must have at least $l$ “valid” candidate values for every private attribute, there must be enough tuples in each l-candidate equivalent set so that $k \geq l$.

Thirdly, we introduce a key technique that be adopted in the process of constructing ESTT - Data Obfuscation, which transforms the original data to random data [60]. It is widely used in protecting data privacy, for example, in [61], they added a random noise to the victim tuple so that the true value is disguised. Ideally, under the limitation of set size $k$ and requirement of l-candidate values, if we can find enough tuples that their private attributes value are mutually different, or at least $|C^B_j| \geq l, \forall j \in \{1, \ldots, m'\}$, we put them together to construct a $E_v$. However, we can not avoid the circumstances that the $\exists |C^B_j| < l$ probably because $V^B_j$ lacks diversity or in the process of constructing last few $E_v$. Under these circumstances, in order to maintain $|C^B_j| \geq l, \forall j \in \{1, \ldots, m'\}$ for every $E_v$,
otherwise privacy guarantee cannot be achieved, we use the Data Obfuscation to conceal original private attribute values by assigning random values. Specifically, regarding to a $E_v$, assign $l$ different values to $v_1[B_j], \ldots, v_k[B_j]$ respectively so that eventually $|C^B_j| \geq l$ in this $E_v$. One can know that, for any $C^B_j$, it is better to keep every true value appearing in $v_1[B_j], \ldots, v_k[B_j]$, then try to randomly pick other non-repeated value(s) that satisfy the constrain of $S_A \Rightarrow B_j$. In the end, every private attribute looks real for the adversaries and also satisfies the requirement of $l$-candidate.

**Implementation:** We now provide the implementation of the algorithm for constructing privacy-preserving equivalent sets with true tuples for a given database $D$ and two input variables $l$ and $k$. The algorithm is straightforward - we start with partitioning $D$ into small groups based on public attributes. Then constructing $l$-candidate equivalent set with $k$-tuples recursively among each partition by adding unprotected tuples until the size of set reaches $k$. Specifically, for a given $E_v$, where $E_v = \{v_1, \ldots, v_k\}$, if there are at least $l$ “valid” values existing in every private attribute, no more action is needed. However, if the number of “valid” values for any private attribute is smaller than $l$, the Data Obfuscation on corresponding private attributes is enforced. Thirdly, we mark all tuples that in the $E_v$ as protected. Continuing constructing the $l$-candidate equivalent set with $k$-tuples until no more $E_v$ with the size of $k$ can be constructed. Gather the remaining unprotected tuples into an equivalent set and enforce Data Obfuscation for any private attribute which can not hold the privacy guarantee. Repeat the process of constructing $E_v$ in each partition. In the end, each $E_v$ achieves $l$-candidate condition with $k$-tuples. We give detail in Algorithm 5.

For example, the construction of 2-candidate equivalent set with 2-tuples, that $E_v = \{v_1, v_2\}$, where $v_1[B_j] \neq v_2[B_j], j \in \{1, 2, \ldots, m'\}$. We start with partitioning data into different partitions based on public attributes. Then construct equivalent set by selecting two true tuples among the partition. If necessary, enforce Data Obfuscation on private attributes where both tuples share the same attribute value. Then mark these two tuples as protected. Continue constructing equivalent sets until no more equivalent set can be constructed. In
Algorithm 5: \(l\)-candidate equivalent set with \(k\)-tuples

Input: \(l, k, D, m, m'\)

Output: \(E_t\)

1 if \(l > \min(V_i^B)\) or \(l > k\) then
2 return \text{FALSE};
3 else
4 Partition\((D, m)\);
5 for \(i = 0; i < (\text{Count}_{\text{Partition}(D,m)})\) do
6 \text{Pick}(v_1, \cdots, v_k \in \text{Partition}_i(D, m));
7 for \(i = 0; i < m'\) do
8 if \(\text{Lcandidate}(v_1[B_i], \cdots, v_k[B_i]) \geq l\) then
9 return \((v_1[B_i], \cdots, v_k[B_i]);
10 else
11 \text{Obfuscation}(v_1[B_i], \cdots, v_k[B_i]);
12 if \(\text{Lcandidate}(v_1[B_i], \cdots, v_k[B_i]) \geq l\) then
13 return \((v_1[B_i], \cdots, v_k[B_i]);
14 end
15 end
16 end
17 return \(E_t\);
18 end
19 end
the end, for each partition group, either no more tuple need to be protected so that we can jump to next partition, or one tuple \( v \) is still unprotected then we need to enforce the Data Obfuscation on \( v \). Recursively execute the constructing process until every tuples in \( D \) are marked as protected. Next, we analyze its utility loss, and provide an optimal solution.

### 4.3.3 Utility Optimization

In Section 4.2.5 we give the measurement of \( U \). In [57], an optimal algorithm for constructing the equivalent sets with virtual tuples is given with minimum \( U \), where the minimum \( U \) can always be found since the algorithm keeps constructing and screening virtual tuples based on query workload \( W \). It uses the Equation (4.9), that given input \( W \), minimize

\[
\sum_{i=1}^{\mid W \mid} |\text{Rank}(v|q_i) - \text{Rank}'(v|q_i)|.
\]

\[
U = \sum_{j=1}^{n} \left( \sum_{i=1}^{\mid W \mid} |\text{Rank}(v_j|q_i) - \text{Rank}'(v_j|q_i)| + \sum_{i=\mid W \mid}^{n} |\text{Rank}(v_j|q_i) - \text{Rank}'(v_j|q_i)| \right) \quad (4.9)
\]

However, minimizing \( U \) is not a practical optimal algorithm here. Firstly, one can observe that optimize \( \sum_{i=1}^{\mid W \mid} |\text{Rank}(v_j|q_i) - \text{Rank}'(v_j|q_i)| \) in Equation (4.9) is difficult. Because we enforce the data obfuscation in constructing the equivalent sets, the \( \text{Rank}'(v_j|q_i) \) is dynamically changing when different values assigned to \( v[B_j] \). Secondly, one can observe that fundamentally, Algorithm 5 is looking for equivalent tuples that have as many of the same private attributes as possible regarding to query workload \( W \), and put them into the same equivalent set. It indeed reduce the utility loss for given \( W \), however, for the rest of \( n - \mid W \mid \) possible queries, the utility loss actually increases.

One can know that the expected global utility loss can be represented as Equation (4.10). For all possible \( n \) queries, the \( \text{Exp}(U) \) is the same. However, data obfuscation brings information loss which leads to decrease in the functionality of score function \( s(t|q) \). Therefore, without considering query workload and maximizing the functionality of score
function \( s(t|q) \), in this subsection, we introduce a better measurement for the utility loss of 1-candidate equivalent set with k-tuples, which is the number of data obfuscation.

\[
\text{Exp}(U) = \frac{1}{n} \sum_{j=1}^{n} \sum_{i=1}^{n} |\text{Rank}(t_j|q_i) - \text{Rank}'(t_j|q_i)| 
\] (4.10)

In real web databases, minimizing the total number of data obfuscation is an optimization because it reduces information loss while protecting privacy. Here we define the utility optimization problem of 1-candidate equivalent set with k-tuples as follow:

**Definition 6.** The optimization problem of l-candidate equivalent set with k-tuples is to find the solution that obfuscates the fewest number of private attribute values.

For l-candidate equivalent set with k-tuples \( E_v, k \geq l \), and \( l \leq \min(V^B_j) \) for any \( j \in \{1, \ldots, m'\} \), we construct \( E_v \) which satisfy the the fewest number of data obfuscation on private attribute values. We prove that the optimization problem of 2-candidate equivalent set with 2-tuples is a NP-complete problem.

**Lemma 4.3.1.** Minimum Length Hamiltonian Circuit problem is NP-complete.

![Figure 4.1: A Hamiltonian Circuit in B](image)

*Proof.* Let \( \text{Same}(v_\alpha, v_\beta) \) be the number of same attribute values between \( v_\alpha \) and \( v_\beta \). Suppose
we have \( n \) tuples in \( D \), and choose a partition which contains \( e \) tuples that have the same public attributes. We can get the following matrix table:

\[
\begin{array}{cccccc}
 & v_1 & v_2 & v_3 & v_4 & \ldots & v_e \\
v_1 & 0 & \text{Same}(v_1,v_2) & \text{Same}(v_1,v_3) & \text{Same}(v_1,v_4) & \ldots & \text{Same}(v_1,v_e) \\
v_2 & \text{Same}(v_1,v_2) & 0 & \text{Same}(v_2,v_3) & \text{Same}(v_2,v_4) & \ldots & \text{Same}(v_2,v_e) \\
v_3 & \text{Same}(v_1,v_3) & \text{Same}(v_2,v_3) & 0 & \text{Same}(v_3,v_4) & \ldots & \text{Same}(v_3,v_e) \\
v_4 & \text{Same}(v_1,v_4) & \text{Same}(v_2,v_4) & \text{Same}(v_3,v_4) & 0 & \ldots & \text{Same}(v_4,v_e) \\
\vdots & \text{Same}(\ldots) & \text{Same}(\ldots) & \text{Same}(\ldots) & \text{Same}(\ldots) & \ldots & \ldots \\
v_e & \text{Same}(v_1,v_e) & \text{Same}(v_2,v_e) & \text{Same}(v_3,v_e) & \text{Same}(v_4,v_e) & \ldots & 0 \\
\end{array}
\]

Table 4.1: Vertex matrix

Based on Table 4.1, we can construct an undirected graph \( G \), where the vertex is the tuple \( v_i \) where \( i \in \{1,2,\ldots,e\} \) and the edge weights are \( \text{Same}(v_\alpha,v_\beta) \) from the matrix. As we can know, \( G \) is a complete graph that every pair of distinct vertices is connected by a unique edge. Figure 4.1A is an example based on 4 tuples in a \( P \) to construct a complete graph. Therefore, \( G \) exists Hamiltonian Circuit. However, according to Lemma 4.3.1, finding the Minimum Length Hamiltonian Path is NP-complete. In Figure 4.1B, the red line path shows a Hamiltonian Circuit.

**Lemma 4.3.2.** Minimum Length Hamiltonian Circuit \( \leq_P \) Optimization problem of 2-candidate equivalent set with 2-tuples

We now proof optimization problem of 2-candidate equivalent set with 2-tuples is NP-complete. Without loss of generality, we select a partition \( P \) where \( \{v_1,v_2,\ldots,v_e\} \in P \), as well as \( v_1,v_2,\ldots,v_e \) share the same public attributes. Next, we reduced the optimization problem of 2-candidate equivalent set with 2-tuples to Minimum Length Hamiltonian Path. In the optimization problem of 2-candidate equivalent set with 2-tuples, intuitively, we pair every two tuples and the goal is to minimize the total weights. Since we have the Minimum Length Hamiltonian Path, which visits each vertex exactly once, we can find a polynomial
time function \( f \), that constructs the optimization solution of 2-candidate equivalent set with 2-tuples by removing edges from the Minimum Length Hamiltonian Path. Therefore, Minimum Length Hamiltonian Path \( \leq_p \) Optimization problem of 2-candidate equivalent set with 2-tuples.

Heuristic algorithm

We have shown that the optimization problem of 2-candidate equivalent set with 2-tuples is NP-complete. We thereby propose a heuristic algorithm to construct 2-candidate equivalent set with 2-tuples. The heuristic algorithm is a slight modification of Algorithm 5 - we make a slight modification on the function of Pick by applying the greedy algorithm. We call it Sorted Weight Algorithm.

We give the detail of Sorted Weight Algorithm as follow: we start with partitioning \( D \) into small groups based on public attributes. Without loss of generality, we assume that each partition has \( n' \) tuples. Among each partition, compute \( \text{Same}(t_\alpha, t_\beta) \), where \( \alpha \neq \beta \). Store \( \text{Same}(t_\alpha, t_\beta) \) to the corresponding cell in the matrix as shown in Table 4.1. Sort the values in matrix by increasing weight. Then start to construct 2-candidate equivalent set with 2-tuples by recursively picking tuples that have the smallest weight edges. If there are at least \( l \) “valid” values exist in every private attribute regarding to picked two tuples, no more action is needed. However, if the number of “valid” values for any private attribute is smaller than \( l \), the Data Obfuscation on corresponding private attributes is enforced. Then we mark these two tuples as “protected” and decrease their weights regarding to other tuples to \(+\infty\). Continue constructing the \( l \)-candidate equivalent set with \( k \)-tuples until no more \( E_v \) with the size of \( k \) can be constructed. Gather the remaining unprotected tuples into an equivalent set and enforce Data Obfuscation for any private attribute which cannot hold the privacy guarantee. Repeat the process of constructing \( E_v \) in each partition. In the end, each \( E_v \) achieves 2-candidate condition with 2-tuples. The time complexity of Sorted Weight Algorithm is \( O(n) \).
Chapter 5: Experimental Results

5.1 Privacy Implications of Time-series data

5.1.1 Experimental Setup

**Devices:** The wearable device we used in the research is one of the most popular devices on the market. It tracks a user’s everyday activities, such as steps, distance, calories burned, etc., and offers multiple ways for a user to upload activity data to its server (for visual analysis, syncing with third-party apps, sharing with friends, etc.) - e.g., through Bluetooth sync with a mobile phone or a wireless sync dongle (which comes pre-packaged with the device) with a computer. A user can view various statistics of sensed activity data online.

**Datasets:** The manufacturer provides a Partner API for developers to access the activity data collected by a user, with granularity as short as one minute. We collect the input sensed sequence $S$ by calling the read method of this API by the end of the day (i.e., the length of each sensed sequence is $60 \times 24 = 1440$). To generate the query sequence, i.e., the event to be detected from the sensed sequence, we consider the following three events:

(a) A 1-minute event depicting a user walking from office to a nearby coffee shop (as shown in Figure 5.1)

(b) A 4-minutes event depicting a user walking from office to a nearby grocery store (as shown in Figure 5.2)

(c) A 7-minutes event depicting a user walking from home to the nearby gym (as shown in Figure 5.3)

**Algorithms Evaluated:** We tested two algorithms discussed in this paper: approximate matching and vertical shift approximate.
For each event, we conducted 20 tests, each corresponding to a different sensed sequence $S$ but the same query sequence $Q$ (corresponding to the event). Out of the 20 sensed sequences, 10 contain the corresponding event while the other 10 does not. Note that each of the 10 sensed sequences containing the event is generated by a separate experiment - i.e., the subsequence corresponding to the event is subject to the inherent variation caused by pedometer errors, human behavior variations, etc.

For the two algorithms, approximate matching and vertical shift approximate, we implemented them in nodeJS and ran the implementation on a machine with 2.2GHz quad-core Intel Core i7 processor, 16GB RAM, and OS X Yosemite operating system. The implementations take as input one parameter: the pre-determined threshold (i.e., upper bound) on the Euclidean distance between the query sequence $Q$ and a sliding window of the sensed sequence $S$. We varied the threshold $\varepsilon$ between 0 and 99 in the tests, with results corresponding to the different thresholds presented in the next section.

5.1.2 Experimental Results

To measure the performance of the inference algorithms, we consider two types of errors:

- **Type-1 Error**, i.e., false negative, is corresponding to cases where the sensed sequence contains the event but the tested algorithm outputs false, i.e., fails to detect the event, for a given value of $\varepsilon$;

- **Type-2 Error**, i.e., false positive, is corresponding to cases where the sensed sequence does not contain the event, yet the tested algorithm outputs true, i.e., makes a wrong detection, for a given value of $\varepsilon$.

We observe that when the query sequence had a shorter length, the number of **Type-1 Error** relatively decreased faster. Meanwhile, when the query sequence had a shorter length, the number of **Type-2 Error** relatively increased faster. For example, while using the approximate matching algorithm, in event (a), the number of **Type-1 Error** decreased from
Figure 5.1: Number of errors when detect event (a) in 20 tests

Figure 5.2: Number of errors when detect event (b) in 20 tests
Figure 5.3: Number of errors when detect event (c) in 20 tests
3 to 0 during $\varepsilon$ increased from 1 to 2. Meanwhile, the number of Type-2 Error increased from 3 to 10 during $\varepsilon$ increased from 0 to 9 (Figure 5.1). In event (b), the number of Type-1 Error decreased from 9 to 0 during $\varepsilon$ increased from 0 to 28. Meanwhile, the number of Type-2 Error increased from 0 to 10 during $\varepsilon$ increased from 0 to 76 (Figure 5.2). In event (c), the number of Type-1 Error decreased from 9 to 0 during $\varepsilon$ increased from 0 to 39. Meanwhile, the number of Type-2 Error increased from 0 to 9 during $\varepsilon$ increased from 0 to 93 (Figure 5.3).

We also observe that when the query sequence had a shorter length, the Type-1 Error happened unlikely in all 10 whole-day time-series tracking-measure sequences which did not contain the query sequence. For example, when $\varepsilon = 0$, in event (a), the Type-1 Error happened in only 3 sensed sequences which did not contain the query sequence, in event (b), the Type-1 Error happened in 9 sensed sequences which did not contain the query sequence, and in event (c), the Type-1 Error happened in all 10 sensed sequences which did not contain the query sequence.

Other than the length of the query sequence, we will discuss in the next section about other factors that affect the result of both algorithm.

In this section, we discuss the various factors that may affect the performance of the inference algorithms. Note that these factors are important not only for understanding the degree of privacy leakage through sharing pedometer readings of wearable devices, but for designing defensive mechanisms as well. For example, as we shall show below, the inference precision is often positively correlated with length of the query sequence. As such, a defensive mechanism can be to increase the time interval of data reporting (e.g., from 1 reading per minute (Figure 5.4) to 1 reading per 2 minutes (Figure 5.5), then to 1 reading per 3 minutes (Figure 5.6)) - so as to reduce the length of the query sequence and thereby the accuracy of inference algorithms.

Specifically, we consider four factors as follows:

- **Length of the query sequence**: An important determining factor for the performance
Figure 5.4: 1 reading per minute

Figure 5.5: 1 reading per 2 minutes
Figure 5.6: 1 reading per 3 minutes
of the inference algorithm is the length of the query sequence $Q$. Generally speaking, the shorter $Q$ is, the more difficult it becomes to reliably detect the event. For example, one can see the above experimental results (e.g., Figure 5.1) that, compared with the other events, Event (a), i.e., walking to a nearby coffee shop, is the most difficult to detect because of its shortest length. On the other hand, the longest sequence (c) yields the best inference precision.

- **Variance of the query sequence** Another factor that affects the accuracy of inference is the inherent uncertainty of the query sequence - i.e., the variance between different sensed sequences of the same event. For example, if a user often conducts Event (a) (i.e., walking to coffee shop) with very different speeds (sometimes casually while other times in a rush), the detection naturally becomes more difficult.

- **Fluctuation of the query sequence** : An orthogonal dimension for depicting the query sequence is the fluctuation within the query sequence - i.e., whether pedometer readings in the query sequence vary significantly. In exact contrary to the variance case, here the higher the fluctuation the more likely inference will succeed. To understand why, consider an example of detecting the event of staying in front of an office desk. Such an event might not see any fluctuation at all in the corresponding query sequence, making it extremely difficult to detect (e.g., to distinguish from the sleeping event). On the other hand, one can see intuitively that the large fluctuation corresponding to events such as working out in a gym provides it with a distinct signature, making it easy to detect.

- **Fluctuation of the (background) sensed sequence** : Finally, the fluctuation of the background sensed sequence also has a significant bearing on the inference accuracy. To understand why, consider two sensed sequences, one representing a user who sits through most of the day and only walk out to perform Event (c), while the other sequence represents a user who remains active all day long, within which Event (c) is
only a small part. Clearly, the first type of sequences likely lead to no false positive, leading to a higher inference accuracy.

Intuitively, it may appear that vertical shift approximate matching algorithm can be more flexible on detecting events, what we can observe from experiment results is that it actually yields worse performance than basic approximate matching algorithm, because it introduces more false positives. As such, we conclude that the basic approximate matching algorithm is performed better in practice.

5.2 Privacy Implications of Hotel Bookings

5.2.1 Experimental Setup

Hardware and Platform: Our experiments were ran on a machine with 2.2GHz quad-core Intel Core i7 processor, 16GB RAM, and running macOS Mojave operating system. For the two algorithms, the baseline matching algorithm and the IHF-matching algorithm are implemented in Python.

Dataset: The dataset we tested in this paper was from one of the most popular OTAs, and it was provided to the OTA’s partner or/and public to launch the related tourism business. We have full access to this dataset and full control of the searching function applied. The dataset is from Expedia, consists of detailed information of 165K properties in the world, and 34,643 of which were located in the United States. Each property has 78 attributes. We also crawl the geographic data from one of the most popular OTAs’ website, which consists area information of the 10 most popular cities to visit in the United States [62].

The public and private attributes were chosen from available attributes. According to the settings in one of the most popular OTAs, we picked 17 attributes as public attributes.

Performance Measure: As discussed, we measure the accuracy of the inference attack. The inference success rate is defined as:
where $n$ is the number of hotels that can be inferred, and $N$ is the total number of hotels in a specific region.

5.2.2 Experimental Results

**Evaluation of Attack Success Guess Rate:** Figure 5.8 shows the probability of success guess over the dataset. We observe that among major cities in the United States, the average inference success rate is over 90%. We must point out that, in some cases even we cannot distinguish two hotels because of they have the identical score, there is actually no difference between two hotels in the perspective of consumers. Because both hotels have identical amenities, it is no doubt that consumers can have same experience no matter which hotel they decide to book. The cause of this kind of situation, after we carefully investigate, is that rival would to run a similar property to gain the max revenue in the same area, or the
5.3 Privacy-preserving framework

5.3.1 Experimental Setup

All our experiments were performed on Mac machine with 2.2GHz quad-core Intel Core i7 processor, 16GB RAM, and OS X Mojave operating system. The algorithm were implemented in Python.

**Dataset:** We used a real-world dataset from eHarmony to verify the utility and efficiency of our privacy-preserving framework. eHarmony dataset contains 486,464 tuples and 53 attributes, of which more than 30 are boolean. In the experiment of computing attack success guess rates, we picked 5 attributes as users’ public attributes as well as picked...
other 5 attributes as users’ private attributes. The respective domain size of public attributes are 5, 2, 2, 5, 6, and the respective domain size of private attributes are 5, 2, 2, 5, 6. In the experiment of measuring performance, among all available attributes, we randomly picked 20 attributes, and designate $m = 10$ of them as public and $m' = 10$ as private. After removal of duplicate tuples, we sampled $n = 300,000$ tuples without replacement as our testing bed. By default, we constructed Query Workload of size 10 by randomly picking 10 tuples from the sample. And we used the ranking function from section 2 with all weights set to 1. **Performance measures:** As explained in 4.3 our framework provides a certain degree of guarantee to privacy while minimize the total utility loss. In this section, we measure privacy by the probability of success guess in Rank Inference Attack [12] and utility loss by one criterion: average top-k rank difference.

5.3.2 Experimental Results

**Evaluation of Attack Success Guess Rate:** Figure 5.9 shows the probability of success guess over the dataset without protection, with ESVT and with ESTT. Theoretically, the adversary can achieve 100% success guess rate by using inference attack. In the case of attacking on the dataset with ESVT and ESTT, the success guess rates are around 50%. It is because the adversary can retrieve two possible values for a private attribute from an equivalent set and cannot identify the true value from them. We can observe that with ESTT, the success guess rate is lower than 50%, it is because the victim may be applied suppression when constructing ESTT.

**Evaluation of utility versus k:** We first investigated the performance of our algorithms for different values of k. Figure 5.10 shows the average top-k utility loss of our algorithms with true tuples(true-tuple), virtual tuples(virtual-tuple) and a baseline algorithm(random) which constructs equivalent sets with randomly picked tuples. As expected, the average top-k utility loss of the baseline method is around 0.5 given diverse k values. The average utility loss of both virtual-tuple and true-tuple are lower than that of the baseline method,
which indicates that our heuristic algorithms can reduce the utility loss while preserving privacy. We also observe that when the value of \( k \) increases, the Utility Loss will show a trend of first increase and then decrease. The reason is that as the number of tuples increases in top-\( k \) results, it is likely that tuples that originally in the top-\( k \) results would have better chance to stay in the new top-\( k \) results after applying algorithms.

**Utility loss versus \( m \) and \( m' \):** The first chart in Figure 5.11 and Figure 5.12 demonstrate the impact of query workload on average top-\( k \) utility loss of our algorithms with virtual tuples and true tuples respectively. As expected, the size of query workload do not have any significant impact on the utility since the tuples in the query workload are randomly generated.

**Utility loss versus database size:** The second chart in Figure 5.11 and Figure 5.12 show the impact of query workload on average top-\( k \) utility loss of our algorithms with virtual tuples and true tuples respectively. As expected, the size of database do not have any significant impact on the utility when \( k \) (10 by default) is much smaller than the size of database.

**Utility loss versus domain size of \( m \):** We then investigate the utility loss under different numbers of public attributes with a fixed number of private attributes and under different numbers of private attributes with a fixed number of public attributes. The third chart in Figure 5.11 and Figure 5.12 shows the result produced by our algorithms with virtual tuples and with true tuples respectively. In the case of the framework with true tuples, the utility will decrease as increasing size of public attributes. When the size of public attributes increases, the number of tuples that have same public attributes would decrease. Therefore it is unlikely to decrease the suppression because of less choices under the circumstances of limited number of candidate true tuples. In the case of the framework with virtual tuples, the utility loss increases with more private attributes and decreases with more public attributes. It is due to the fact that with higher proportion of public attributes, tuples in the same equivalent set would have a higher proportion of common attribute values which leads to less difference in score.
Utility loss versus weight ratio: In this experiment, we fixed the weight of all private attributes to 1 and varied the weights of all public attributes from 1 to 3. The last chart in Figure 5.11 and Figure 5.12 show that when weight ratio increases, the utility loss will decrease. The reason is that when the weight ratio increases, the suppression and alternations applied to private attributes would have less influence to the score. Therefore, more tuples that originally in the top-\(k\) results would have less rank differences after applying our algorithms.

Figure 5.9: Attack Guess Success Rate
Figure 5.10: Virtual tuples (True tuples) vs random

Figure 5.11: Utility loss of virtual tuples

Figure 5.12: Utility loss of true tuples
Chapter 6: Conclusion

In the dissertation, we initiated three problems.

Firstly, we initiated the study of a novel problem on the privacy disclosure, specifically the inference of sensitive human behavior events, from data sensed by wearable devices such as activity trackers. Through real-world experiments, we found that the pedometer readings captured and shared by an off-the-shelf activity tracker indeed enables the accurate inference of events such as grocery shopping, walking to coffee shop, visiting gym, etc. We also discussed the various factors that may affect the accuracy of inference algorithms and in turn can be leveraged for the design of defensive mechanisms.

Secondly, we demonstrated a novel privacy issue that hotel’s hidden information is compromised under the inference attack to the online hotel booking system. Through real-world experiments, we found that current hotel booking products indeed break the asymmetry model and potentially would decrease the revenue.

Thirdly, we addressed the issue related to privacy-preserving in ranked retrieval model, which has been adopted widely to web databases. We proposed a privacy-preserving framework to protect private attributes privacy which can defend not only the rank-based inference attack but also arbitrary attacks. We introduced a classification of adversaries and their capability. For domain-ignorant adversaries, we have designed ESVT and proved its privacy guarantee. For domain-expert adversaries, we designed ESTT and proved its privacy guarantee. For ESTT, we developed heuristic algorithm for practical situations under the consideration of minimizing the utility loss. We showed simple and efficient solutions can be developed to deal with the privacy disclosure in ranked retrieval model with little utility loss.
Bibliography


[59] Jiawei Han Yongjian Fu Wei Wang, Jenny Chiang Wan Gong Krzysztof Koperski, Deyi Li, Yijun Lu Amynmohamed Rajan Nebojsa Stefanovic, and Betty Xia Osmar R Zaiane. Dbminer: A system for mining knowledge in large relational databases. In *Proc. Intl. Conf. on Data Mining and Knowledge Discovery (KDD ’96)*, pages 250–255, 1996.


