IT-University of Copenhagen

Masters Thesis

Data Anonymization

Author:
Balazs Toth

Supervisor:
Carsten Schürmann

A thesis submitted in fulfilment of the requirements for the degree of Msc. in Software Development

July 14, 2015

IT University of Copenhagen

Abstract

Msc. in Software Development

Data Anonymization

by Balazs Toth

Several generalization and permutation based anonymization methods have been published recently that claimed to be effective in protecting against membership disclosure. I show with the help of a proposed algorithm that permutation in itself is not effective in protecting against membership disclosure, if it is supposed that the adversary has access to an external public table P, and private table $T \subseteq P$. I also present an algorithm that combines permutation with generalization. Permutation combined with generalization is effective in protecting against membership disclosure, but my proposed algorithm results in bigger information loss, and bigger bucket sizes, than multidimensional generalization. The algorithms are evaluated in the context of a real-world scenario.

Contents

List of Figures

Chapter 1

Introduction

1.1 Introduction

The success of data mining, machine learning and data analysis is indisputable. The success of these techniques rely on the availability of high quality and high volume data and effective information sharing, therefore there is huge demand for all kinds of data: financial data, health data, internet transaction, clickstream data, and so on... In the other hand, it is irresponsible to publish datasets containing sensitive information without anonymization, because adversaries might misuse it.

Today, the primary privacy protection practice mainly relies on policies and guidelines defined by the Health Insurance Portability and Accountability Act (HIPAA)[1] in the USA or The European Community Directive 95/46/EC [2] in the EU. These policies and guidelines restrict the types of publishable data. There are many critiques of these policies and guidelines [3] [4] [5]. The limitation of this approach is that it either distorts the dataset excessively, and still doesn't provide enough privacy or requires a trust level that is impractically high in many data-sharing scenarios. Furthermore, policies cannot prevent adversaries not to misuse the data, and contracts and agreements cannot assure that sensitive data will not be negligently misplaced and end up in the hands of adversaries. For the reasons mentioned above, there is a great need to develop methods and tools for anonymizing data. The goal of data anonymization is to preserve individual privacy while leaving the anonymized data practically useful. There has been a lot of research in this area recently. This can be viewed as a technical response to complement the privacy policies [5].

In the recent works [6] [7] [8] [5], many kind of privacy models were proposed. In this thesis, I am going to focus on the table linkage model. In the table linkage model, a

privacy violation happens, if the adversary is able to determine with high certainty that an individual's record exists in a published database. This kind of privacy violation is called membership disclosure. For example, let's suppose that a dataset containing patients with a particular type of cancer is published for research purposes. Determining with high certainty that an individual is present in this medical dataset could be damaging for the individual. This individual might not get hired by a company, because the company identifies her/him in the dataset, or she/he might not receive health insurance.

In this thesis, I am going to examine the effectiveness of generalization and permutation based data anonymizition techniques in protecting against membership disclosure. Comparing generalization based techniques to differential privacy is also the subject of future work. I am going to focus on a table linkage model that supposes that the adversary has access to an external public table P, and private table $T \subseteq P$ [6]. The main contribution of this thesis that it shows that (1) permutation based techniques are not efficient in protecting against membership disclosure, and (2) if a published dataset has to be protected against membership disclosure, generalization based techniques are recommended for anonymization. I am going to evaluate my thesis with experiments.

1.2 Structure and contribution

In chapter 2, I review the basic concepts of generalization based data anonymization with special focus on protection against membership disclosure.

In chapter 3, I review the permutation based anonymization techniques with special focus on protection against membership disclosure.

In chapter 4, I am going to show (and later I am going to verify it with experiments) that (1) permutation in itself is not effective in protecting against membership disclosure, (2) and some degree of generalization is unavoidable in order to effectively protect against membership disclosure. This is the main contribution of my thesis.

In chapter 5, I introduce a technique that combines permutation with generalization, and I compare this technique to generalization. I also propose an algorithm that combines permutation and generalization and effectively protects against membership disclosure. (1) I am going to prove, if this algorithm is used with MAPLM as partitioning algorithm, permutation is useless, and permutation combined with generalization cannot achieve better result regarding discernibility metric value/ bucket sizes, than MPALM in any cases. (2) Moreover, I am going to show an example, where this algorithm using K-Mondrian as partitioning algorithm, reveals more information, than MPALM.

In chapter 6, I conduct three experiments. In the first experiment, I evaluate the effectiveness of a permutation based algorithm in protecting against membership disclosure. In the second experiment, I evaluate the effectiveness of algorithm 6 regarding discernibility metric and amount of generalization, as compared to MPALM. In the third experiment, I examine: (1) how the amount of generalization changes, and (2) how the running time changes, as k is increased in algorithm 6.

Chapter 2

Hiding the Presence of Individuals with generalization

In this chapter, I am going to review the basic concepts of generalization based anonymization.

2.1 Review of the basic concepts

Name	Zip	Age	Salary
Clara	5370	25	30k
Julie	5370	25	15k
Ada	5370	27	30k
Sara	5371	27	40k
Bob	5371	27	42k
Taylor	5372	25	25k
Ashley	5372	25	45k
Amanda	5373	25	9k

FIGURE 2.1: 1st private table

Name	Zip	Age
Elizabeth	5370	25
Clara	5370	25
Julie	5370	25
Ada	5370	27
Sara	5371	27
Bob	5371	27
Taylor	5372	25
Ashley	5372	25
Amanda	5373	25
Jesper	5372	27

FIGURE 2.2: 1st public table corresponding to figure 2.1

2.1.1 Attribute types

An attribute of a dataset can be identifier, quasi-identifier(QI) and sensitive attribute. Identifier attributes, such as name, social security number, Mobil number can explicitly identify the owner of the record. E.g. the only identifier attribute in table 2.1 is the

Zip	Age	Salary
537*	27	30k
537*	27	40k
537*	27	12k
537*	25	30k
537*	25	15k
537*	25	25k
537*	25	45k
537*	25	9k

FIGURE 2.3: full domain anonymization of table 2.1

Zip	Age	Salary
537*	27	30k
537*	27	40k
537*	27	12k
537*	25	30k
537*	25	15k
537*	25	25k
5370	25	45k
5370	25	9k

FIGURE 2.4: multidimensional generalization of table 2.1 with global rewriting, k=2

Name	Zip	Age	Salary
Julie	5370	21	15k
Sara	5371	27	40k
James	5372	32	42k
Barbara	5376	39	50k

FIGURE 2.5: 2nd private table

Name	Zip	Age
Julie	5370	21
Ada	5370	27
Sara	5371	27
James	5372	32
Mary	5376	38
Barbara	5376	39

FIGURE 2.6: 2nd public table corresponding to figure 2.5

Zip	Age
537*	31-40
537*	31-40
537*	31-40
537*	21-30
537*	21-30
537*	21-30

FIGURE 2.7: full domain generalization of public table 2.6, k=2

Zip	Age
537*	31-40
537*	31-40
537*	31-40
537*	21-30
537*	21-30
537*	21-30

FIGURE 2.8: multidimensional generalization of public table 2.6 with global rewriting, k=2

name attribute. A quasi-identifier (QI) attribute such as year of birth, gender, zip code, native country, or a combination of quasi-identifier attributes with external information can potentially re-identify individual records[9]. E.g. QI attributes in table 2.1 are zip and age. Note that salary could be also a QI attribute, but for simplicity, I suppose here that salary is only a sensitive attribute. Sensitive attributes contain information that considered to be sensitive, such as disease, salary. E.g. a sensitive attribute in table 2.1 is salary. An attribute can be both quasi-identifier and sensitive.

2.1.2 Attack types

There are two big categories of attack types. Record linkage, attribute linkage, and table linkage are in the first category. An attack of this category is considered to be successful, if an adversary manages to link the victim's record to a sensitive attribute in a published table, or to find out, if the victim is in the published table. In these attack types, it is assumed that the adversary has background knowledge of the victim - she or he knows the QI attribute values of the victim. In record and attribute linkages, it is also assumed that the adversary knows that the victim's record is in the published table. In table linkage model, it is supposed that the adversary doesn't know, if the victim's record is in the published database or not, and she or he tries to determine the presence or absence of it.

The probabilistic attack is in the second category. An attack of this category is considered to be successful, if the adversary has a large difference between the prior and posterior beliefs, after analyzing the published table [8].

2.1.3 Generalization and Specialization of QI attributes

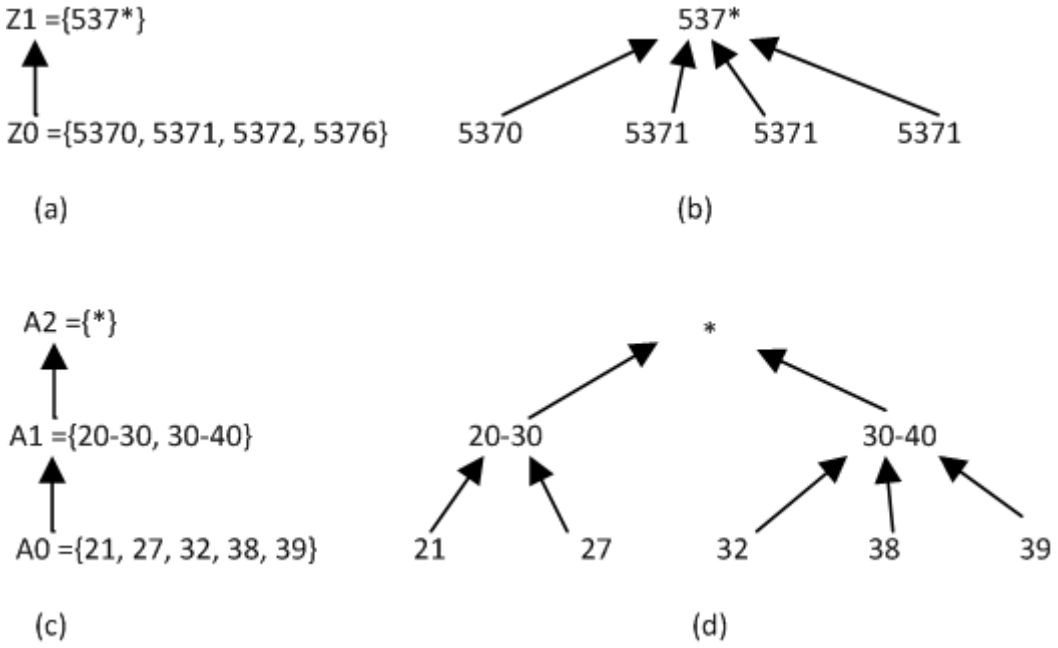

FIGURE 2.9: Domain generalization hierarchies for Zip code (a, b) and Age (c, d)

It is not enough just to remove the identifier attributes in a dataset that is going to be published. It has been shown that 87% of the population in the United States can be uniquely identified using the birth date, sex, and zip code attributes [7]. The more QI attributes a dataset contain, the more individuals can be identified with high certainty.

In order to prevent record linkage attacks, Samarati and Sweeney [9] [10] introduced the anonymization techniques of generalization and suppression of QI attributes. The basic idea is to replace a specific value to a more general value of that specific value. For example, continuous attribute domains can be generalized into ranges, the n last digits of zip codes can be dropped, taxonomy trees can be created for categorical attribute domains. Examples for zip code suppression, and generalization of age are shown in table 2.4 and 2.8.

Generalization and suppression can use domain generalization hierarchies. For generalization based algorithms, domain generalization hierarchies are the only way to deal with categorical attributes. LeFevre at al. formalized the concept of domain generalization hierarchies [11]. They defined a domain generalization relationship by <D, and they use the notation D_i <D D_j to denote that domain D_j is either identical to or a domain generalization of D_i. For two domains D_i and D_j, the relationship D_i <D D_j indicates that the values in domain D_j are the generalizations of the values in domain D_i. More precisely, a many-to-one value generalization function is associated with each domain generalization D_i <D D_j. LeFevre at al. define domain generalization hierarchy as a set of domains that is totally ordered by the relationship <D. Examples of domain generalization hierarchies are shown in Figure 2.9.

Here, I am going to formalize one step table specialization and table specialization with respecting the notation that Negrgiz at al use [6]: In a table T , T[c][r] refers to the value of column c, row r. T[c] refers to the projection of column c on T and T[.][r] refers to selection of row r on T. I am going to use the definitions of specialization function and table specialization in the further parts of this thesis. Many existing algorithms that enforces some kind of privacy model, use specialization.

DEFINITION 1 (ONE STEP GENERALIZATION FUNCTION)

Given a data value v, a *one step generalization function* Ψ returns the next value in the totally ordered set of all generalizations of v.

Here, I consider only specializations with domain generalization hierarchies. For example, let the totally ordered set of generalizations S = {Denmark, N.Europe, Europe, * }. In this case Ψ(Denmark) = N.Europe, Ψ(N.Europe) = Europe and Ψ(Europe) = *.

DEFINITION 2 (ONE STEP SPECIALIZATION FUNCTION)

Given a data value v, a *one step specialization function* Υ returns the next value in the totally ordered set of all specializations of v.

For example, let the totally ordered set of specializations S = {*, Europe, N.Europe, Denmark}. In this case Υ(*) = Europe, Υ(Europe) = N.Europe and Υ(N.Europe) = Denmark.

v_1 one step generalization of $v_2 \iff v_2$ one step specialization of v_1

DEFINITION 3 (ONE STEP TABLE SPECIALIZATION)

Given two tables T_1 and T_2, I say T_2 is a *one step specialization* of T_1 if and only if $|T_1| = |T_2|$ and records in T_1, T_2 can be ordered in such a way that T_2[x][j] = $\Upsilon(T_1$[x][j]) for every possible index j, and x $\in$ QI attributes, and T_2[i][j] = T_1[i][j] for every attribute i different from x.

Note, that one step table specialization is invertible, which means that it is always possible to reconstruct T_1 from T_2. Therefore inversion is also transitive. T_2 can be reconstructed to T_1 by applying $\Psi(T_2$[x][j]) for every possible index j. Note also that specialization is invertible, but generalization is not. This is because in the former, more information is added, whereas in the latter, some information is lost.

Example: Table 2.11 is a one step specialization of Table 2.10. Table 2.10 can be reconstructed from Table 2.11 by applying Ψ on the Zip column for every tuple of Table 2.11.

T_1 one step generalization of $T_2 \iff T_2$ one step specialization of T_1.

DEFINITION 4 (TABLE SPECIALIZATION)

Table specialization is a transitive closure of one step table specializations.

Note, that table specialization is also invertible. If T_3 is a one step specialization of T_2, and T_2 is a one step specialization of T_1, then T_3 is a specialization of T_1. T_3 can be reconstructed to T_2, and T_2 can be reconstructed to T_1, therefore T_3 can be reconstructed to T_1.

Example: Table 2.14 is a specialization of Table 2.10. Table 2.11 can be reconstructed from Table 2.14 by applying Ψ on the Age column for every tuple of Table 2.14. Table 2.10 can be reconstructed from Table 2.11 by applying Ψ on the Zip column for every tuple of Table 2.11.

Negrgiz at al. [6] formalized the concept of the equivalence class in the following way:

DEFINITION 5 (EQUIVALENCE CLASS)

The *equivalence class* of tuple t in dataset T^* is the set of all tuples in T^* with identical quasi-identifiers to t.

Example: in Table 2.2, the equivalence class for tuple Clara is {Elizabeth, Clara, Julie}.

In some of the privacy models [6], [12], it is supposed that the adversary has access to all publicly known data (represented in a public table P) that links names to other set of attributes (e.g., date of birth, gender, race). When a data holder publishes a table with sensitive information, the adversary can match QI attributes in both tables to determine unique links between records in the public and published table [6], [12].

2.1.4 k-anonymity, l-diversity

Samarati and Sweeney also introduced k-anonymity in order to prevent record linkage attacks [7][9][10]. Table T is considered to be k-anonymous, if the size of every equivalence class eq is at least k: $\forall$ eq $\in$ T: |eq| $\geq$ k. In a k-anonymous table, every tuple is indistinguishable from at least k -1 other tuples with respect to QI attributes, therefore the probability of linking an individual to a specific tuple through QI attributes is at most 1/k[5].

K-anonymity cannot prevent attribute linkage attacks in all cases. For example, there are cases where there are k different records in an equivalence class, but all k records have the same sensitive value, thus the adversary can learn the victim's sensitive value by attribute linkage attack. In order to prevent attribute linkage attacks, Machanavajjhala et al. introduced the diversity principle, called l-diversity[8]. The l-diversity requires each equivalence class (or bucket) to contain at least l sensitive values. There are several variations of this principle. For example one of them is called (k, e)-anonymity, and it constraints the minimum difference that sensitive values have to have in an equivalence class (or in a bucket)[13].

2.1.5 Full-domain generalization and Local Recoding with Multidimensional generalization

2.1.5.1 Full domain generalization

There are several models for producing a generalized table T' from table T. Full-domain generalization is one of them. It was introduced by Samarati and Sweeney [7][9]. It is a single dimensional global-recoding scheme. In single-dimensional generalization, some function is defined: $f_i : D_{Q_i} \rightarrow$ D' for each QI attribute. A generalization T' of T is obtained by applying each f_i to the values of QI_i in each tuple of T [11]. In global rewriting, if an attribute is generalized, all values of that attribute are generalized to the same level of domain generalization hierarchy inside a bucket. For example, in Table

2.3 , if zip code '5371' is suppressed to '537*', then it also requires suppressing '5372' to '537*'. The search space for full domain generalization is smaller than the search space for other schemes, therefore optimal solution can be found with respect to a cost metric. The advantage of full domain generalization is that it produces an homogeneous set of values, and the disadvantage of it is that it generalizes too much, and results in big information loss. Examples of full domain generalization are shown in Table 2.3 and 2.7.

2.1.5.2 Local recoding

In local recoding, some instances of an attribute may be less generalized while other instances are more generalized inside a bucket. For example, in an equivalence class, one instance of the zipcode '5371' can be suppressed to 537*, while an other instance of the zipcode '5371' can remain unsuppressed. Local recoding schemes result in less information loss, than global recoding schemes, but the resulted anonymized dataset is not homogeneous, therefore it can have less data utility, than global recoding schemes in some cases[5].

2.1.5.3 Multidimensional generalization

Multidimensional generalization is defined by a single function f : $(D_{Q_1} \times ... \times D_{Q_n})$ $\rightarrow$ D', which is used to recode the domain of value vectors associated with the set of QI attributes. Generalization T' of T is obtained by applying f to the vector of QI values in each record of T [11]. This scheme allows attribute values to be independently generalized into different parent groups of the domain generalization hierarchies in different equivalence classes. For example, zip code can be suppressed in one of the equivalence classes, while in an other equivalence class, it can remain unsuppressed, like in Tabe 2.4. This scheme results in less information loss, than single domain generalization schemes. Note, in this multidimensional scheme, all records in an equivalence class are generalized to the same levels of generalizations, but local recoding does not always have such constraint [5]. Note also, that multidimensional generalization can be combined with global or local rewriting [11][14].

2.1.6 Algorithms that enforce k-anonymity

2.1.6.1 Full domain generalization

Optimal k-anonymization, for a given cost metric, is feasible for small data sets with full-domain generalization, since the search space for the full-domain generalization scheme is much smaller, than for other schemes [11][5]. The state of art algorithms that enforces k-anonymity with optimal full domain generalization are Incognito[11] and K-Optimize[15].

2.1.6.2 Multidimensional generalization

LeFevre et al. introduced K-Mondrian, a greedy top-down specialization algorithm for finding a minimal k-anonymization with multidimensional generalization scheme[14]. K-Mondrian performs a specialization on an equivalence class, if each of its specialized equivalence classes contain at least k records. In most of the cases, K-Mondrian results in less information loss, than any single dimensional generalization based algorithm. The trade-off is that its anonymous result set is not homogeneous.

K-Mondrian can be used with global rewriting and with a special kind of local rewriting. If K-Mondrian is used with strict multidimensional partitioning, then it uses global rewriting, and results in non-overlapping equivalence classes. If K-Mondrian is used with relaxed multidimensional partitioning, then it uses a version of local rewriting, and results in potentially overlapping equivalence classes.

K-Mondrian algorithm, as shown in algorithm 1, can handle numeric attributes with or without the presence of domain generalization hierarchies. Both strict partitioning and relaxed partitioning can be applied on numeric attributes. Although, K-Mondrian can handle categorical attributes only with the presence of domain generalization hierarchies, and only strict partitioning can be applied on categorical attributes in most of the cases. Examples of result sets of the K-Mondrian algorithm as shown in table 2.4 and 2.8.

2.2 Table linkage model

There can be situations where the adversary is uncertain of the existence of an individual's record in the anonymized database. If the adversary determines the existence or absence of an individual's record in the anonymized dataset, a privacy violation happens. In some cases, hiding presence of individuals in private databases are crucial. For example, if a dataset contains only diabetics, and the adversary determines the presence of an individual in this dataset, then he/she learns that the disclosed individual

Algorithm 1 K-Mondrian with strict multidimensional partitioning [14]

```
1: function ANONYMIZE(partition)
2:     if (no allowable multidimensional cut for partition) then
3:         return φ : partition → summary
4:     else
5:         dim ← choose_dimension()
6:         fs ← frequency_set(parrtition, dim)
7:         splitVal ← find_median(fs)
8:         lhs ← {t ∈ partition : t.dim ≤ splitVal}
9:         rhs ← {t ∈ partition : t.dim > splitVal}
10:        return Anonymize(rhs) ∪ Anonymize(lhs)
```

has diabetes. Even if the adversary doesn't determine precisely the presence or absence of an individual in the anonymized database, his/her estimate of the probability that the target individual has diabetes, can increase or decrease significantly after he/she examines the anonymized database.

Nergiz et al. proposed a table linkage model where they suppose that the adversary has access to an external public table P, which is the super set of the private table T (P $\supseteq$ T)[6]. They bound the probability of inferring the presence of any potential victim's record within a range $\delta_t = (\delta_{min}, \delta_{max})$, where δ_t is the probability that the record of individual t exists in T. Nergiz et al. formalized δ-presence in the following way[6]:

DEFINITION 6 (δ-PRESENCE)

Given an external public table P, and a private table T, we say that δ-presence holds for a generalization T^* of T, with δ=(δ_{min},δ_{max}) if

$\delta_{min} \leq P(t \in T \mid T^*) \leq \delta_{max} \; \forall \; t \in P$

In such a dataset, we say that each tuple t $\in$ P is *δ-present* in T. Therefore, δ = (δ_{min},δ_{max}) is a range of acceptable probabilities for P(t $\in$ T | T^*). And it is assumed that T $\subseteq$ P.

Example: the probability (δ_{Sara}) that Sara's record (zip: 5371, age: 27) exists in Table 2.3 is $\frac{3}{4}$, because there are 3 records in the private table, and there are 4 records in the public table that correspond to Sara's QI values.

In the other hand, there is an other kind of table linkage model [16] [17] that doesn't suppose that the adversary has access to an external public table P, which is the super set of the private table T. I am going to discuss it in the next chapter.

2.2.1 Algorithms that enforces δ-presence

It is possible to obtain δ-present tables with k-anonymity or l-diversity based algorithms, if weak k-anonymity[12] or weak l-diversity is used, since weak k-anonymity take both a public table and a private table into consideration. Moreover, there is no direct relationship between k or l and δ, therefore in order to obtain a δ-present result set, k-anonymity or l-diversity based algorithms generalize much more, than algorithms that concentrate only on obtaining δ-present private tables[6].

Nergiz et al.[6] present two algorithms that enforce δ-presence, and outperform k-anonymity or l-diversity based algorithms in obtaining δ-present tables: One of them is SPALM, an optimal δ-presence-anonymization, for a given cost metric, with full-domain generalization, and the other one is MPALM, a greedy top-down specialization algorithm for finding a minimal δ-presence with multidimensional generalization scheme. I now concentrate on MPALM, as shown in algorithm 2, which is a variation of the Mondrian algorithm[14]. MPALM distinguishes from K-Mondrian in the way that the splitting criteria is the δ-presence property, instead of the k-anonymity.

Nergiz et al.[6] show also that finding optimal solutions with MPALM may be infeasible, but sub-optimal multidimensional generalizations are likely to result in less information loss than optimal single-dimension generalization.

Example: MPALM takes table 2.1 as private table and 2.2 as public table. The δ_{min} and δ_{max} are set to 0.1 and 0.9 respectively. In the first step (2.10), the fully generalized version of Table 2.1 is added to the queue Q. In the second step (2.11), MPALM specializes on the Zip attribute, and it examines, if it is possible to split the bucket by any value of the Zip column without breaking δ-presence, but it is not possible. In the third step (2.12), MPALM specializes on the Age attribute, and it examines, if it is possible to split the bucket by any value of the Age column without breaking δ-presence, and it is possible. The bucket is split by value 25 (2.12), and the two new buckets are added to the queue Q. In the 5th step and the 6th step (2.14), MPALM specializes on the Zip attribute, and it examines, if it is possible to split the buckets by any value of the Zip column without breaking δ-presence, but it is not possible. The buckets that are shown in 2.13 are added to the result set of MPALM.

Algorithm 2 MPALM [6]

Require: publicly available table P; private table T
Ensure: return a $(\delta_{min}, \delta_{max})$-present multidimensional generalization ot T w.r.t P

Let Q be an empty queue of tables;
Let R be an empty set of tables;
enqueu(Q, P);
while Q is not empty **do**
    $T_i \leftarrow dequeue(Q)$
    $j \leftarrow 0$
    repeat
        j++;
        $C \leftarrow choose_{column}(T_i, T, j)$
        $v \leftarrow choose_{value}(T_i, T, C)$
    until v $\neq$ null or j = number_of_columns(T)
    if v $\neq$ null {split column C on value v} **then**
        $T_i^{<v} \leftarrow \{t \in T_i \mid \text{t[C]} < v\}$
        $T_i^{\geq v} \leftarrow \{t \in T_i \mid \text{t[C]} \geq v\}$
        enqueue(Q, $T_i^{<v}$);
        enqueue(Q, $T_i^{\geq v}$);
    else{ no allowable δ-present split found }
        $R \leftarrow R \cup T_i$
for all $T_i \in R$ **do**
    return smallest bounding box of $T_i \cap T$;

Zip	Age	IsInT
537*	20-30	0
537*	20-30	1
537*	20-30	1
537*	20-30	1
537*	20-30	1
537*	20-30	1
537*	20-30	1
537*	20-30	1
537*	20-30	1
537*	20-30	0

FIGURE 2.10: step 1 of mpalm, private table: 2.1, public table: 2.2

Zip	Age	IsInT
5370	20-30	0
5370	20-30	1
5370	20-30	1
5372	20-30	1
5372	20-30	1
5373	20-30	1
5370	20-30	1
5371	20-30	1
5371	20-30	1
5372	20-30	0

FIGURE 2.11: step 2 of mpalm, private table: 2.1, public table: 2.2

Zip	Age	IsInT
537*	25	0
537*	25	1
537*	25	1
537*	25	1
537*	25	1
537*	25	1
537*	27	1
537*	27	1
537*	27	1
537*	27	0

FIGURE 2.12: step 3 of mpalm, private table: 2.1, public table: 2.2

Zip	Age	IsInT
537*	25	0
537*	25	1
537*	25	1
537*	25	1
537*	25	1
537*	25	1
537*	27	1
537*	27	1
537*	27	1
537*	27	0

FIGURE 2.13: step 4 of mpalm, private table: 2.1, public table: 2.2

Zip	Age	IsInT
5370	25	0
5370	25	1
5370	25	1
5372	25	1
5372	25	1
5373	25	1
5370	27	1
5371	27	1
5371	27	1
5372	27	0

FIGURE 2.14: step 5 of mpalm, private table: 2.1, public table: 2.2

Chapter 3

Hiding presence of Individuals with permutation

In this chapter, I am going to review the permutation based anonymization techniques. Some of the articles discussing permutation based algorithms use a table linkage model that doesn't suppose that the adversary has access to an external public table P that is the super set of the private table T, and they argue that permutation based techniques can effectively protect against membership disclosure in this interpretation.

3.1 Permutation based techniques

Algorithms applying permutation make use of generalization based algorithms for partitioning the tuples of an input dataset into buckets. The difference between generalization and permutation based algorithms is that permutation based algorithms don't modify or generalize the values inside their buckets, but randomly permute the values within each column inside every bucket, thus de-associate the relationship among some of the attributes.

3.1.1 Bucketization

In bucketization, random permutation operations are applied inside the buckets only within the columns that contain the sensitive values, as shown in figure 3.1 and 3.2. For example in Figure 3.2 in the first bucket, the values 40k, 15k are randomly permuted. The QI attribute values are not permuted; they are preserved in their original form. Since bucketization doesn't modify or permute the QI attribute values, it is easy to find

out, if an individual's record exists or not in the database anonymized with bucketization. It can't be applied for data that don't have a clear separation between QI attributes and sensitive attributes. In the other hand, if only the l-diversity constraint matters, it provides better data utility, than generalization based techniques [13]. It doesn't provide an effective protection against membership disclosure in none of the table linkage model interpretations, since it doesn't de-associate the relationship among QI attributes.

Zip	Age	Salary
5370	25	9k
5370	25	45k
5372	25	25k
5372	25	15k
5373	25	30k
5370	27	42k
5371	27	30k
5371	27	40k

FIGURE 3.1: bucketized table of 2.1

Zip	Age	Salary
5370	21	40k
5371	27	15k
5372	32	50k
5376	39	42k

FIGURE 3.2: bucketized table of 2.5

3.1.2 One-attribute-per-column-slicing and Anatomy with permutation

3.1.2.1 One-attribute-per-column-slicing

Likewise in bucketization based algorithms, in one-attribute-per-column-slicing, tuples are partitioned into buckets with the help of generalization based algorithms, but instead of generalization, random permutation operation is applied on the values within each column within every bucket, as shown in Figure 3.3 and 3.4. The difference between bucketization and one-attribute-per-column-slicing is that in one-attribute-per-column-slicing, QI attribute values are also randomly permuted in order to break the linking among the columns containing the QI attributes. For example, in Figure 3.4 in the first bucket, the values 5370, 5371 are randomly permuted, and the values 27, 21 are also randomly permuted.

3.1.2.2 Anatomy with permutation

The technique called anatomy in its original version doesn't apply permutation on QI attributes, but like bucketization it provides better data-utility than generalization based techniques, if only the l-diversity constraint matters [17].

On the other hand, anatomy can be combined with permutation, and in this way it is very similar to the one-attribute-per-column slicing. It partitions the tuples into buckets

Zip	Age	Salary
5370	25	9k
5370	25	45k
5372	25	25k
5372	25	15k
5373	25	30k
5370	27	42k
5371	27	30k
5371	27	40k

FIGURE 3.3: One-attribute-per-column slicing of 2.1 with Mondrian strict partitioning

Zip	Age	Salary
5370	27	15k
5371	21	40k
5372	39	42k
5376	32	50k

FIGURE 3.4: One-attribute-per-column slicing of 2.5 with Mondrian strict partitioning

in the same way as one-attribute-per-column slicing, and permutation operation can be applied on the QI attributes inside the buckets as in one-attribute-per-column slicing. The difference between anatomy with permutation and one-attribute-per-column is that anatomy with permutation distributes the QI attributes (PQT e.g. 3.5) and the sensitive attributes (PST e.g. 3.6) in two separate tables as shown in Figure 3.5 - 3.6 and 3.7 - 3.8. The two tables can be joined by the group id's.

Zip	Age	Group id
5373	25	1
5370	25	1
5372	25	1
5372	25	1
5370	25	1
5371	27	2
5370	27	2
5371	27	2

FIGURE 3.5: PQT of 2.1

Group id	Salary
1	9k
1	45k
1	25k
1	15k
1	30k
2	42k
2	30k
2	40k

FIGURE 3.6: PST of 2.1

Zip	Age	Group id
5370	21	1
5371	27	1
5372	32	2
5376	39	2

FIGURE 3.7: PQT of 2.5

Group id	Salary
1	40k
1	15k
2	50k
2	42k

FIGURE 3.8: PST of 2.5

Xianmang et al. [18] argued that anatomy combined with permutation protects against membership disclosure. Note that they didn't suppose that the adversary has access to a external public table P, and private table T $\subseteq$ P. They came up the following lemma:

From an adversary's perspective, the probability (γ_t) to find out whether an individual t exists in the QI-group QI_j by the presence attack is at most $\gamma_t = \frac{\prod_i^d |n_{i_j}|}{|QI_j|^d}$, where n_{i_j} denotes the number of the value $t.A_i$ on attribute A_i in QI_j.

Zip	(Age, Salary)
5370	(27, 40k)
5370	(27, 30k)
5370	(25, 15K)
5371	(25, 30k)
5371	(25, 9K)
5372	(25, 25k)
5372	(25, 45k)
5373	(27, 42k)

FIGURE 3.9: sliced version of 2.1 with Mondrian relaxed partitioning

Zip	(Age, Salary)
5370	(27, 40k)
5371	(21, 15k)
5372	(39, 50k)
5376	(32, 42k)

FIGURE 3.10: sliced version of 2.5 with Mondrian relaxed partitioning

Example of the previous lemma: Suppose the victim is Sara {Zip: 5371, age: 27}, and her age and Zip are available to the adversaries. From the adversary's perspective the probability that Sara's record exists in bucket$_1$ of table 3.7 is $\delta_{Sara}= \frac{1}{2} \times \frac{1}{2} = \frac{1}{4}$.

3.1.3 Slicing

In slicing, datasets are partitioned both horizontally and vertically, as shown in Figure 3.9 and 3.10. Vertical partitioning means that attributes are grouped into columns based on the correlations among the attributes. If some columns have high correlation, they are partitioned together. In my example in Figure 3.9 and 3.10, the Age and Salary attributes are partitioned into a group due to high correlation. Tiancheng Li et al.[16] showed that slicing can handle high dimensional data more efficiently than other methods due to the vertical partitioning.

Horizontal partitioning means partitioning the tuples into buckets similarly as in bucketization and one-attribute-per-column slicing. After the partioning, the generalization phase is skipped, and random permutation operations are applied instead within every bucket in every column likewise in bucketization and one-attribute-per-column slicing. For example in 3.9 in the first bucket, the values (27, 40k),(27, 30k),(25, 15K),(25, 30k) are randomly permuted, and the values 5370, 5370, 5370, 5371 are also randomly permuted.

3.1.3.1 A slicing algorithm

Tiancheng Li et al. [16] describe a slicing algorithm, and they argue that slicing prevents membership disclosure. Their proposed algorithm is shown in Algorithm 3 and 4. In this algorithm, attribute partitioning is done by clustering based on column correlation. Pearson correlation coefficient and mean-square contingency coefficient are used for measuring correlations, and k-medoid algorithm is used for clustering.

Algorithm 3 Algorithm tuple-partition(T, l) [16]

```
1: Q = {T}; SB = ∅.
2: while Q is not empty do
3:     remove the first bucket B from Q; Q = Q - {B}.
4:     split B into two buckets B_1 and B_2, as in Mondrian.
5:     if diversity-check(T, Q ∪ {B_1, B_2} ∪ SB, l) then
6:         Q = Q ∪ {B_1, B_2}.
7:     else SB = SB ∪ B.
8:     return SB.
```

Algorithm 4 Algorithm diversity-check(T,T^*, l) [16]

```
1: for all tuple t ∈ T do
2:     L[t] = ∅.
3: for all bucket B in T* do
4:     record f(v) for each column value v in bucket B.
5:     for all tuple t ∈ T do
6:         calculate p(t,B) and find D(t, B).
7:         L[t] = L[t] ∪ {(p(t, B), D(t, B))}.
8: for all tuple t ∈ T do
9:     calculate p(t, s) for each s based on L[t].
10:    if p(t, s) ≥ 1/l, return false.
11: return true.
```

In the tuple partitioning phase of the algorithm, a modified version of the Mondrian algorithm [14] is used. The Mondrian algorithm is applied with relaxed partitioning. Relaxed partitioning of the Mondrian algorithm produces potentially overlapping buckets as mentioned earlier in the first chapter. A new kind of l-diversity constraint is introduced by Tiancheng Li et al.[16], and this l-diversity is the tuple partitioning splitting constraint in the algorithm: Tuple t satisfies l-diversity iff for any sensitive value s, $p(t, s) \leq 1/l$. A sliced table satisfies l-diversity iff every tuple in it satisfies l-diversity.

Tiancheng Li et al.[16] argue that slicing, and especially their proposed algorithm protects against membership disclosure. In their argumentation, it is not supposed that the adversary has access to an external public table P, and private table T $\supseteq$ P. They introduce two new two measures for membership disclosure protection. The first measure is the number of fake tuples, and the second measure is to consider the number of matching

buckets for original tuples and that for fake tuples. They define these measures in the following way: Let D be the set of tuples in the original data and let $\overline{D}$ be the set of tuples that are not in the original data. Let D^S be the sliced data. In their interpretation, the goal of membership disclosure is to determine whether tuple t $\in$ D or t $\in \overline{D}$. If t $\in$ D, t has at least one matching bucket in D^S. Let a tuple be an original tuple if it is in D. Let a tuple be a fake tuple if it is in $\overline{D}$ and it has at least one matching bucket in D^S. Their first measure is the number of fake tuples. They argue that if the number of fake tuples is 0, the membership of every tuple in D^S can be determined with high certainty. According to their second measure, if the number of matching buckets for original tuples and that for fake tuples are similar enough, membership information of every tuple in D^S is protected because the adversary cannot distinguish original tuples from fake tuples.

Chapter 4

Determining the membership of individuals' records in permuted tables

In the last chapter I reviewed some articles that claim that permutation based techniques can efficiently protect against membership disclosure. In these articles, it was not supposed that the adversary has access to an external public table P that is the super set of the private table T. In this chapter, I am going to show (and I will verify it in the Experiments chapter) that (1) permutation in itself is not effective in protecting against membership disclosure, if another table linkage model is used in which it is supposed that the adversary has access to a external public table P, and private table T $\subseteq$ P, (2) and some degree of generalization is unavoidable in order to effectively protect against membership disclosure. To decide which table linkage model is more usable is the subject of future work.

4.1 Determining the membership of individuals' records in permuted tables

In this section, I am going to use a table linkage model which supposes that the adversary has access to an external public table P, and private table T $\subseteq$ P. I show that the above discussed permutation based anonymization techniques are not effective in protecting against membership disclosure by showing that it is possible to determine the membership of some individuals' records in private table T that is anonymized with the above discussed permutation techniques. I consider two cases: (1) when a

private table T is anonymized with one-attribute-per-column-slicing, and T results in non-overlapping buckets, and (2) when T is anonymized with slicing and T results in overlapping buckets. Here, I will not consider the case of bucketization and anatomy with permutation, because bucketization doesn't provide an effective protection against membership disclosure in any of the table linkage model interpretations, and anatomy with permutation is very similar to one-attribute-per-column-slicing, therefore it doesn't need its own example.

Here, I extend definition 4, the δ-presence definition that was originally formalized by Ngeriz et al. [6]:

DEFINITION 6 (δ-PRESENCE)

Given an external public table P, and a private table T, I say that δ-presence holds for a permutation or generalization T^* of T, with δ=($\delta_{\min}$,$\delta_{\max}$) if

$\delta_{\min} \leq \mathrm{P}(\mathrm{t} \in \mathrm{T} \mid T^*) \leq \delta_{\max} \; \forall \; \mathrm{t} \in \mathrm{P}$

In such a dataset, I say that each tuple t $\in$ P is δ-present in T. Therefore, $\delta = (\delta_{\min},\delta_{\max})$ is a range of acceptable probabilities for $\mathrm{P}(\mathrm{t} \in \mathrm{T} \mid T^*)$. And it is assumed that T $\subseteq$ P.

4.1.1 Non-overlapping buckets

First, I consider a case where private table T is anonymized with one-attribute-per-column-slicing, and the tuple partitioning algorithm in one-attribute-per-column-slicing uses K-Mondrian with strict partitioning, therefore one-attribute-per-column-slicing results in non-overlapping buckets, and every individual's record t $\in$ T can potentially belong to only one bucket. I show that in this case, it is possible to determine the membership of some individual's records in T.

In this case, given a private table T anonymized with one-attribute-per-column-slicing, and a public table P, and $T \subseteq P$, and bucket $A \subseteq$ P, and bucket B $\subseteq$ T, and $B \subseteq A$, and individual's record $t \in$ A, from an adversary's perspective, the probability (δ_t) to find out whether t exists in B can be computed by the Algorithm 5:

The algorithm takes two sets and the id of an individual's record t as parameters. Set A is a subset of the public table P, and set B is a subset of the private table T. The value ranges of A are the same as the value ranges of B.

In line 2, the algorithm computes all possible subsets of A that has the same number of elements as B. In line 5-9, it loops through all possible subsets of A. If a subset of A has the same frequency for each value as B, then the possibleOutcomes variable

Algorithm 5 Getδ

1: **function** GETδ(set A, set B, recordId t)
2: $subsets\_of\_A \leftarrow A$
3: $possibleOutcomes \leftarrow 0$
4: $successfulOutcomes \leftarrow 0$
5: **for all** $C \in subsets\_of\_A$ **do**
6: **if** (B and C have the same frequency for each values) **then**
7: $possibleOutcomes \leftarrow possibleOutcomes + 1$
8: **if** $(t \in C)$ **then**
9: $successfulOutcomes \leftarrow successfulOutcomes + 1$
10: **return** $\frac{successfulOutcomes}{possibleOutcomes}$

gets incremented by 1, and if it also contains the id of individual's record t, then the successfulOutcomes variable gets also incremented by 1. In line 10, the algorithm returns the quotient of successfulOutComes and the possibleOutComes variables.

Example: Given public table P shown in 2.6, and Sara's record s ∈ P, and private table T shown in 3.4, and bucket A shown in 4.1, $A \subseteq P$, and bucket B shown in 4.2, and $B \subseteq T$, and $B \subseteq A$. The probability (δ_s) that s is in T is 100%. The steps of algorithm 5 are shown in figure 4.3.

Algorithm 5 has exponential time complexity ($O(2^n)$) due to the step in line 2 that iterates through the possible subsets of A.

Name	Zip	Age
Juli	5370	21
Ada	5370	27
Sara	5371	27

FIGURE 4.1: bucket A from the external table 2.6

Zip	Age
5370	27
5371	21

FIGURE 4.2: bucket B from the one-attribute-per-column sliced table 3.4 or anatomy table 3.7

step	subsets of A	frequencies of A	frequencies of the current subset	possibleOut.	successfulOut.
1	Juli, Ada	5370:1, 5371:1, 21:1, 27:1	5370:2, 21:1, 27:1	0	0
2	Juli, Sara	5370:1, 5371:1, 21:1, 27:1	5370:1, 5371:1, 21:1, 27:1	1	1
3	Ada, Sara	5370:1, 5371:1, 21:1, 27:1	5370:1, 5371:1, 27:2	1	1

FIGURE 4.3: steps of DeltaValue

4.1.2 Overlapping buckets

Secondly, I consider a case where private table T is anonymized with slicing, and the tuple partitioning algorithm in slicing uses K-Mondrian with relaxed partitioning, therefore slicing results in potentially overlapping buckets, and every individual's record t ∈ T can potentially belong to many buckets. I show that in this case, it is possible to determine the membership of some individual's records in T.

In this case, given a private table T anonymized with slicing, and a public table P, where T ⊆ P, and buckets $B_1, ..., B_n \subseteq$ T, and individual's record t ∈ T, and t can potentially $\in B_1, ..., B_n$, and buckets $A_1, ..., A_n \subseteq$ P, and $B_1 \subseteq A_1, ..., B_n \subseteq A_n$, from an adversary's perspective, the probability (δ_t) to find out whether t exists in T can also be computed by the Algorithm 5. Before running the algorithm, set B_{union} has to be created by taking the union of $B_1, ..., B_n$, and set A_{union} has to be created by taking the union of $A_1, ..., A_n$. In this case, Algorithm 5 takes the following parameters: A_{union}, B_{union} and id of t.

Example: Given public table P shown in 2.1, and Sara's record s ∈ P, and private table T shown in 3.9. One of the suppositions of the table linkage models is that the adversary knows Sara's QI attribute values (Zip: 5371, Age: 27). s has 2 matching buckets in T, where s can potentially be: bucket B_1 shown in 4.5 and bucket B_2 shown in 4.7. Buckets $A_1 \subseteq P$, $A_2 \subseteq P$, $B_1 \subseteq A_1$ and $B_2 \subseteq A_2$. A_1 and A_2 are shown in 4.4 and 4.6. A_{union} and B_{union} are shown in 4.8 and 4.9. The result of algorithm 5 is that the probability (δ_{Sara}) that Sara is in T is 100%.

Name	Zip	Age
Elizabeth	5370	25
Clara	5370	25
Julie	5370	25
Ada	5370	27
Sara	5371	27
Bob	5371	27

FIGURE 4.4: bucket A_1 from the external table E

Zip	(Age, Salary)
5370	(27, 40k)
5370	(27, 30k)
5370	(25, 15K)
5371	(25, 30k)

FIGURE 4.5: bucket B_1 from the sliced table S

Name	Zip	Age
Sara	5371	27
Bob	5371	27
Taylor	5372	25
Ashley	5372	25
Amanda	5373	25
Jesper	5372	27

FIGURE 4.6: bucket A_2 from the external table E

Zip	(Age, Salary)
5371	(25, 9K)
5372	(25, 25k)
5372	(25, 45k)
5373	(27, 42k)

FIGURE 4.7: bucket B_2 from the sliced table S

Name	Zip	Age
Elizabeth	5370	25
Clara	5370	25
Julie	5370	25
Ada	5370	27
Sara	5371	27
Bob	5371	27
Taylor	5372	25
Ashley	5372	25
Amanda	5373	25
Jesper	5372	27

FIGURE 4.8: bucket A_{union} from the public table P

Zip	(Age, Salary)
5370	(27, 40k)
5370	(27, 30k)
5370	(25, 15K)
5371	(25, 30k)
5371	(25, 9K)
5372	(25, 25k)
5372	(25, 45k)
5373	(27, 42k)

FIGURE 4.9: bucket B_{union} from the private table T

Chapter 5

Permutation combined with generalization

In the previous chapter, I showed that permutation in itself couldn't protect effectively against membership disclosure, if it is supposed that the adversary has access to an external public table P, and private table $T \subseteq P$. In this chapter, I introduce a new technique that combines permutation with generalization, and I compare this technique to generalization. I also propose a new algorithm that combines permutation and generalization and effectively protects against membership disclosure. (1) I am going to prove that, if this algorithm is used with MPALM as its partitioning algorithm, then permutation is useless, and that permutation combined with generalization cannot achieve better result regarding Discernibility Metric, than MPALM in any cases. (2) Furthermore, I am going to show an example, where this algorithm using K-Mondrian as its partitioning algorithm, reveals more information, than MPALM.

5.1 Permutation combined with generalization

Here, I extend again definition 4, the δ-presence definition that was originally formalized by Ngeriz et al. [6]:

DEFINITION 6 (δ-PRESENCE)

Given an external public table P, and a private table T, I say that δ-presence holds for a permutation $T^{*'}$ of generalization T^{*} of T, with δ=($\delta_{\min}$,$\delta_{\max}$) if

$\delta_{\min} \leq \mathrm{P}(\mathrm{t} \in \mathrm{T} \mid T^{*'}) \leq \delta_{\max} \; \forall \; \mathrm{t} \in \mathrm{P}$

In such a dataset, I say that each tuple t ∈ P is δ-present in T. Therefore, $\delta = (\delta_{\min},\delta_{\max})$ is a range of acceptable probabilities for P(t ∈ T | $T^{*'}$). And it is assumed that T ⊆ P.

The algorithm I propose, shown in algorithm 6, combines permutation with generalization. It returns a set of δ-present buckets, and takes the following parameters: a public table, a private table, δ_{min}, δ_{max}, domain generalization hierarchies (DGH's), a generalization based algorithm for partitioning and optionally k or l, if the generalization based algorithm is k-anonymity or l-diversity based. In this part of the thesis, I will focus only on generalization based algorithms applied for partitioning that use multidimensional generalization with global re-coding. Checking (δ_{min}, δ_{max}) presence on anonymizations using local re-coding and containing overlapping equivalence classes is much more difficult and complex.

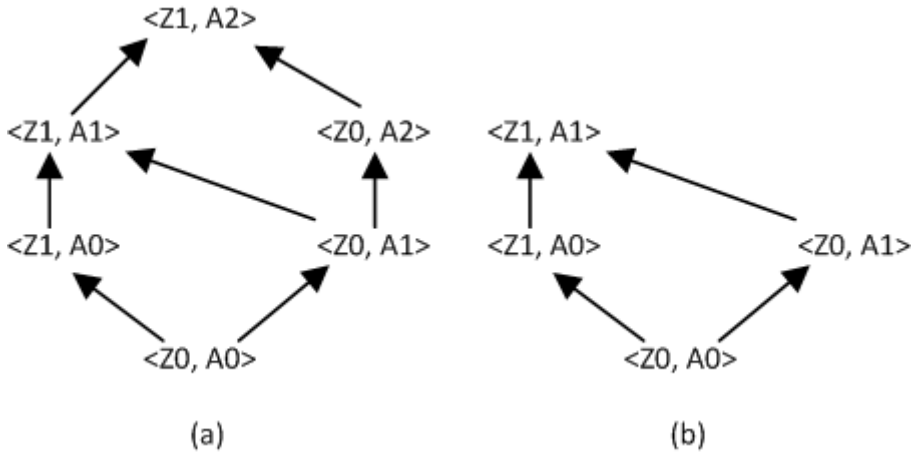

FIGURE 5.1: Generalization lattice for the Zip code and Age attributes (a) and the node <Z1, A1> and its children and grandchildren (b)

In line 2 of algorithm 6, the dataset is partitioned into buckets with a chosen generalization based algorithm, but the tuples are not generalized at this point. From line 4 to line 9, the algorithm iterates through every buckets. In line 5, the algorithm generates all of the possible generalization nodes that are the children or grandchildren of the node of the current bucket (including the node of the current bucket) and sorts them in ascending order of the sum of the generalization levels of the nodes. This step needs some more explanation: The domain generalization hierarchies of QI attributes can be combined to form a multi-attribute generalization lattice similarly like in [11]. An example lattice for Zip code and Age created from the domain generalization hierarchies 2.9 is shown in Figure 5.1(a), where the node <Z0, A2> is a direct multi-attribute generalization of <Z0, A1> and an implied multi-attribute generalization of <Z0, A0>. The generalization levels of every bucket is correspondent to a generalization node. For example, <Z1, A1> from Figure 5.1 (a) could be a generalization node for one of the buckets. This node and its children and grandchildren are shown in Figure 5.1 (b). The sorted list of

the sum of the possible generalization levels of this node is the following: {<Z0, A0>, <Z0, A1>, <Z1, A0>, <Z1, A1>}.

In line 6-8, the values of a bucket are getting more and more generalized with the help of the generalization nodes, until the bucket is δ-present after the permutation of the values within each column. Generalization and permutation are applied inside each bucket in this way. They are finally added to the result set, if they are δ-present.

Algorithm 6 CombinedAlg

```
1:  function CombinedAlg(public table, private table, δ_min, δ_mag, DGH's, partitioning algorithm, k)
2:      buckets ← partitioning with partitioning algorithm
3:      resultset ← null
4:      for all bucket ∈ buckets do
5:          sortedGeneralizationNodeSet ← DGH's
6:          for all node ∈ generalizationNodeSet do
7:              while ¬δ − Present(bucket, min, max) do
8:                  bucket ← generalizeAndPermutate(bucket, node)
9:              resultset.Add(bucket)
10:     return resultset
```

Algorithm 6 has exponential time complexity ($O(2^n)$) due to the δ-presence checking.

5.1.1 Choosing the generalization based algorithm for partitioning

5.1.1.1 Using MPALM for partitioning

In this section, I investigate, whether the result set of algorithm 6 can reveal more information, than the result set of MPALM, if algorithm 6 is used with MPALM as its partitioning algorithm.

Among the generalization based algorithms that effectively protect against membership disclosure, MPALM produces the smallest buckets compared to the other existing algorithms, therefore its Discernibility Metric value is the lowest. Furthermore, its Loss Metric value is also the lowest[6]. If an algorithm could specialize some of the attributes and permute the values within each column in any bucket that is in the result set of MPALM without loosing the δ-presence property of the bucket, then that algorithm could have better results, than MPALM, because it would have the same bucket sizes as MPALM, and it would reveal more information, than MPALM. But the following proof shows that is not possible.

DEFINITION 7 (ONE STEP BUCKET PERMUTATION)

Given three buckets B_1, B_2 and B_3, I say B_3 is a *one step permutation* of B_1, if B_2 is a one step specialization of B_1 on x QI attribute, and B_3 is derived form B_2 by permuting the values within the column where the specialization was done in B_2.

Note, that one step bucket permutation is also invertible, which means that it is always possible to reconstruct B_1 from B_3. B_1 can be reconstructed with inversion to B_3 by applying $\Psi(T_3[x][j])$ for every possible index j.

Example: Bucket 5.2 is a one step permutation of Bucket 5.3. Bucket 5.2 can be reconstructed from Table Bucket 5.3 by applying Ψ on the Zip column for every tuple of Table 5.3.

DEFINITION 8 (BUCKET PERMUTATION)

Bucket permutation is a transitive closure of one step bucket permutations.

Note, that bucket permutation is also invertible. If B_3 is a on step permutation of B_2, and B_2 is a one step permutation of B_1, then B_3 is a specialization of B_1. B_3 can be reconstructed with inversion to B_2, and B_2 can be reconstructed with inversion to B_1, therefore B_3 can be reconstructed to B_1.

Example: Bucket 5.8 is a permutation of Bucket 5.6.

DEFINITION 9 (BREAK δ-PRESENCE)

Given an external public table P, and a private table T, a permutation $T^{*'}$ of generalization T^* of T, δ=($\delta_{\min}$,$\delta_{\max}$), and $T \subseteq P$, if

Bucket b *breaks δ-presence* $\iff$ b $\in$ P, and there exists record t $\in$ b that $\neg(\delta_{min} \leq P(t \in T \mid T^{*'}) \leq \delta_{max})$

FACT 1

If MPALM terminates at a result set R, and a one step specializations is applied on a bucket b $\in$ R, b breaks δ-presence.

Lemma 1: Let R be the result set of MPALM. Every permuted bucket pb derived from R breaks δ-presence.

Proof by induction on the sequence of one step bucket permutation steps:

1. Base case: If a one step permuted bucket pb is derived from a bucket b $\in$ result set R of MPALM, only one QI attribute can potentially contain multiple values within a QI column, all of the other QI attributes are single valued. In this case, only the values of one attribute are permuted, therefore the permutation doesn't break any association among the attributes, therefore pb also breaks δ-presence by FACT 1.

2. Inductive step: If a permuted bucket pb is derived from a bucket b $\in$ result set R of MPALM, it can be reconstructed with inversion to a one step permutation of b, and any one step permutation of b breaks δ-presence by Fact 1.

Since both the base case and the inductive step have been performed, by mathematical induction, the statement in Lemma 1 holds for all permuted bucket pb derived from R. □

COROLLARY 1. Specialization combined with permutation cannot be applied on the result set of MPALM without breaking δ-presence.

If algorithm 6 is used with MPALM as partitioning algorithm it cannot reveal more information than MPALM, because of theorem 1. Permutation is useless in this case. And I can also conclude that permutation combined with generalization cannot achieve better result regarding Discernibility Metric value, than MPALM in any cases.

5.1.1.2 Using K-Mondrian for partitioning

It is also possible to partition a dataset into δ-present buckets with K-Mondrian. With the same $\delta_{\min}$, $\delta_{\max}$ values, K-Mondrian results in bigger buckets, than MPALM [6], therefore often, it is possible to create permuted buckets from some of the buckets that are in the result set R of K-Mondrian. Note that the necessary amount of k has to be chosen, in order to get δ-present buckets.

Example steps of Algorithm 6 are shown in table 5.2, 5.3 and 5.4: In this case, algorithm 6 is used with K-Mondrian as its partitioning algorithm and it takes the following parameters: table 2.2 as public table and table 2.1 as the private table, δ_{min}: 0.1 and δ_{max}: 0.9. First, K-Mondrian partitions the table into one bucket (table 5.2), because of the k=4 constraint. After, it permutes the values within each column (table 5.3), and checks, if the permuted bucket is δ-present. But it is not δ-present, therefore it generalizes the bucket according to node <Z1, A0>, and permutes the values within each column (table 5.4). It checks, if the generalized bucket is δ-present, which it is and therefore it adds this bucket to the result set. Since in this case, there were only one bucket, the algorithm terminates.

In some cases, algorithm 6 with K-Mondrian as partitioning algorithm can reveal more information, than MPALM. For example, if dataset T in table 5.5 is anonymized with $\delta_{\min}$: 0.5, $\delta_{\max}$: 0.5, algorithm 6 could reveal both column col1 and col2 of T in a permuted form, as shown in table 5.8, while MPALM could reveal either col1 or col2 of T, but not both, as shown in table 5.6. Note that MPALM has smaller buckets, than algorithm 6 also in this example.

Zip	Age	Salary
5370	25	30k
5370	25	15k
5370	27	30k
5371	27	40k
5371	27	42k
5372	25	25k
5372	25	45k
5373	25	9k

FIGURE 5.2: step 1 of algorithm 6 with K-Mondrian as partitioning algorithm, public table: 2.2, private table: 2.1, $\delta_{\min}$: 0.1 ,$\delta_{\max}$: 0.9

Zip	Age	Salary
5370	27	15k
5371	25	15k
5372	27	30k
5371	25	40k
5372	27	42k
5371	25	25k
5370	25	45k
5370	27	9k

FIGURE 5.3: step 2 of algorithm 6 with K-Mondrian as partitioning algorithm, public table: 2.2, private table: 2.1, $\delta_{\min}$: 0.1 ,$\delta_{\max}$: 0.9

Zip	Age	Salary
537*	27	15k
537*	25	15k
537*	27	30k
537*	25	40k
537*	27	42k
537*	25	25k
537*	25	45k
537*	27	9k

FIGURE 5.4: step 3 of algorithm 6 with K-Mondrian as partitioning algorithm, public table: 2.2, private table: 2.1, $\delta_{\min}$: 0.1 ,$\delta_{\max}$: 0.9

col1	col2	isInT
0	0	0
0	1	1
1	0	1
1	1	0

FIGURE 5.5: dataset T - Here the public table and the private table is merged in one table. IsInT attribute is 0, if the record is not in the private table, isInT attribute is 1, if the record is in the private table)

col1	col2	isInT
0	*	0
0	*	1
1	*	1
1	*	0

FIGURE 5.6: result set of MPALM, public and private tables: 5.5, $\delta_{\min}$: 0.5, $\delta_{\max}$: 0.5

col1	col2	isInT
*	*	0
*	*	1
*	*	1
*	*	0

FIGURE 5.7: result set of K-Mondrian, k=4, public and private tables: 5.5, $\delta_{\min}$: 0.5, $\delta_{\max}$: 0.5

col1	col2	isInT
0	0	0
0	1	1
1	0	1
1	1	0

FIGURE 5.8: result set of algorithm 6 with K-Mondrian as partitioning algorithm, k=4, public and private tables: 5.5, $\delta_{\min}$: 0.5, $\delta_{\max}$: 0.5

Chapter 6

Experiments

I describe three experiments. In the first experiment, I evaluate the efficiency of permutation and generalization based techniques in protecting against membership disclosure. As I mentioned in the introduction, this experiment supports my thesis by demonstrating that permutation based techniques are not efficient in protecting against membership disclosure, and some degree of generalization is unavoidable in order to effectively protect against membership disclosure. In the second experiment, I evaluate the efficiency of algorithm 6 regarding discernibility metric and amount of generalization, as compared to MPALM. This experiment demonstrates that MPALM is more efficient, than algorithm 6. In the third experiment, I examine: (1) how the amount of generalization changes, and (2) how the running time changes, as k is increased in algorithm 6. The third experiment demonstrates that even if algorithm 4 might have better results in some rare cases and in specific aspects, often it is not usable due to its time complexity. The conclusion of this thesis is that if a published dataset has to be protected against membership disclosure, generalization based techniques are recommended for anonymization. The experiments support this conclusion.

I implemented the algorithms used in the experiments in C# using .NET 4.5 framework and an additional external library [19] for computing cartesian products. I ran the experiments on a Dell Insipiron N5110 with 8 GB of memory and Intel Core i7-2670QM CPU @ 2.20 GHz 2.20 GHz.

In all of the experiments, I used a simulated dataset that was created through random selection of a subset of the adult dataset from the UC Irvine Machine Learning Repository [20]. I have chosen this dataset for evaluation, because it has become the standard dataset to compare different kinds of anonymization algorithms (e.g. it is used in [6] [11] [14], [16] [18]). The entire adult dataset consists of 32561 records. I removed all records

	Attribute	Distinct Values	Generalizations
1	Age	74	5-, 10-, 20-, 120- year ranges(5)
2	Gender	2	Suppression(1)
3	Race	5	Suppression(1)
4	Martial Status	7	Taxonomy tree(2)
5	Education	16	Taxonomy tree(3)
6	Native country	41	Taxonomy tree(2)
7	Work class	7	Taxonomy tree(2)
8	Occupation	14	Taxonomy tree(2)
9	Salary class	2	Suppression(1)

FIGURE 6.1: Description of the Adults dataset and the applied domain generalization hierarchies

that contained null values, since my experimental implementation of the algorithms cannot handle null values. After removing the null values, the database consisted of 30161 records. I considered this database as the public table, and I have randomly selected a subset of 2000 records as the dataset of individuals whose discovery in the dataset is to be protected against. In the dataset, I used the following attributes as quasi-identifiers: age, gender, race, work class, salary, martial status, education, occupation, native country as shown in Figure 6.1.

6.1 Determining the membership of individuals' records

In the first experiment, I anonymize the simulated dataset with bucketization, one-attribute-per-column slicing, slicing, K-Mondrian, algorithm 6 and MPALM. In the permutation based algorithms, I use K-Mondrian with strict partitioning as partitioning algorithm. I try to determine, if some individuals' records are present in the anonymized datasets with algorithm 5. In this experiment, a record in the private table is a re-identified record, if its probability to exist in the private table with respect to the public table is 100%. I ran the algorithms with several settings. First, I set k=2, then k=4, ..., k=32. In slicing, I partitioned horizontally the attributes Native country and Race, furthermore Working class, Occupation and Salary class due to their high correlations. Note that MPALM doesn't take k as parameter, therefore I ran MPALM only once.

The results are shown in Figure 6.2. As expected, the percentage of re-identified records is the highest (93.95%) in bucketization, and it doesn't decrease, as the sizes of the buckets increase. The percentages of re-identified records in Slicing and one-attribute-per-column-slicing are almost the same as in bucketization, but as the sizes of buckets increase, the The percentage of re-identified records decreases a bit, but not significantly. It is important to note that as k increases, the data utility preservation decreases [16], [14], [6], [11]. Here, I can conclude that slicing and one-attribute-per-column-slicing is

only a slightly more efficient in protecting against membership disclosure, if the buckets are big. Moreover, if the buckets are small, they are not more efficient than bucketization. Furthermore, one-attribute-per-column-slicing performs a little better, than slicing, if the buckets are big.

K-Mondrian and algorithm 6 can protect efficiently against membership disclosure, if k is big enough. In this case, if k= 4, none of the records can be re-identified in the private table. K-Mondrian and algorithm 6 have the same results in protecting against membership disclosure.

As I mentioned, I ran MPALM only once. MPALM also protects efficiently against membership disclosure, additionally MPALM results in less information loss and smaller buckets, than K-Mondrian [6].

Instead of the k-anonymity constraint, I could have used l-diversity in the Mondrian partitioning algorithm, but it wouldn't have made a difference, because there is not a direct relationship between l and δ [6].

The result demonstrates that permutation in itself is not effective in protecting against membership disclosure, and some degree of generalization is unavoidable in order to effectively protect against membership disclosure.

6.2 Comparing algorithm 6 to MPALM

As the first experiment demonstrated, permutation in itself is not efficient in protecting against membership disclosure. But generalization in itself, and permutation combined with generalization can efficiently protect against membership disclosure.

I have presented algorithm 6 that combines permutation with generalization and enforces δ-presence. In this experiment, I use algorithm 6 with K-Mondrian as its partitioning algorithm.

I now compare algorithm 6 to MPALM on the simulated data in order to find out, whether algorithm 6 can result in less information loss in some of the cases. In the comparison, I use two kind of general-purpose cost metrics: discernibility metric [21] and amount of generalization. Discernibility metric measures the size of the resulted buckets of the algorithms, and charges penalties dependent on the sizes of the buckets. The penalty on bucket b is $|\mathrm{b}|^2$. The overall penalty cost of anonymized table T is given by

$$DM(T) = \sum_{bucket b \in T} |\mathrm{b}|^2$$

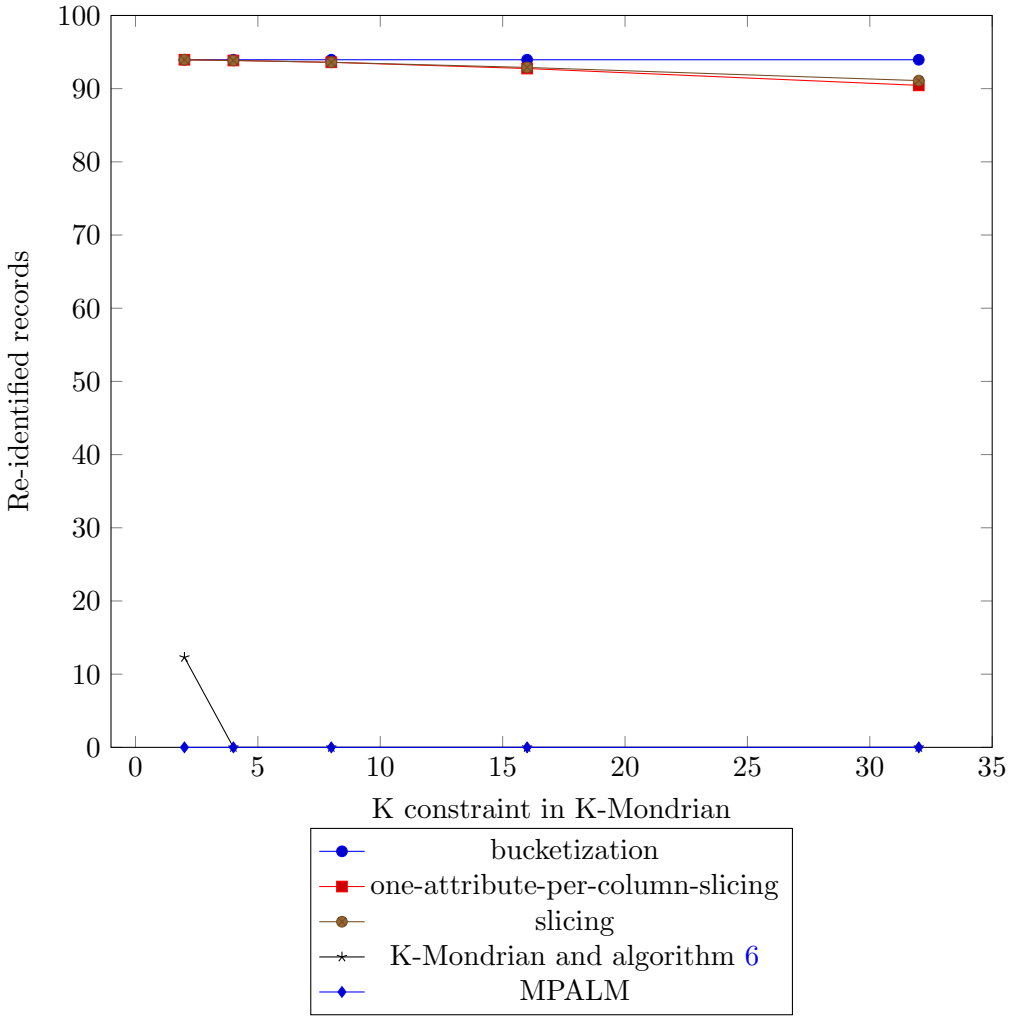

FIGURE 6.2: Re-identification of individuals in the dataset anonymized with a permutation based algorithm

There exists a metric that is called height metric [9]. It measures information loss based on the summation of the generalization levels applied to QI attributes. Originally it was used for generalization based techniques. I use a modified version of this metric, and call it 'amount of generalization'. It measures the information loss of a tuple t by summing up the generalization levels applied to QI attributes of t:

$$\text{AG(t)}= \sum_{generalization_n of t} generalization_i.level$$

The amount of generalization of table T can be computed in the following way:

$$\text{AG(T)}= \sum_{t \in T} AG(t)$$

I ran algorithm 6 and MPALM with many $\delta_{\min}$, $\delta_{\max}$ values. In algorithm 6, I always chose the smallest integer for k that could ensure δ-presence. As expected, algorithm 6 has worse results regarding discernibility metric, than MPALM in each examined

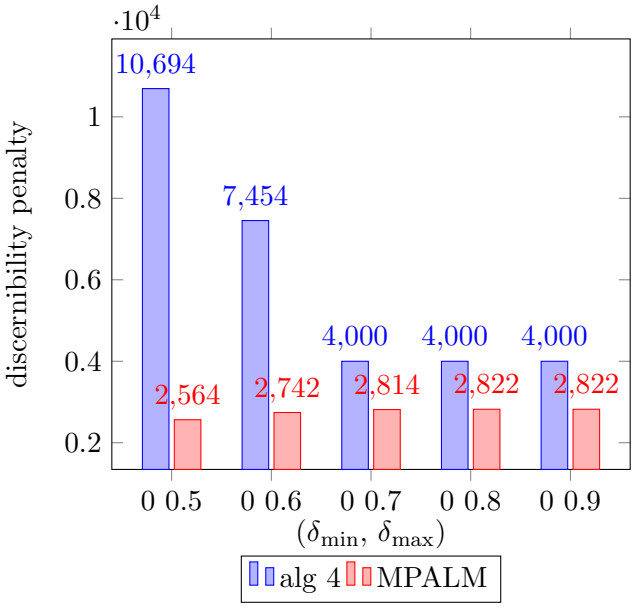

FIGURE 6.3: k recpectively: 12, 8, 4, 4, 4

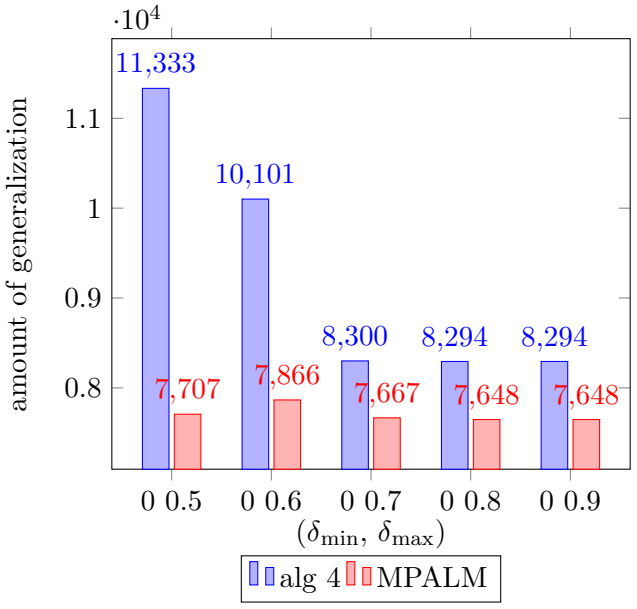

FIGURE 6.4: k recpectively: 12, 8, 4, 4, 4

case. It is due to the fact that algorithm 6 uses K-Mondrian for partitioning, and K-Mondrian needs to have bigger buckets, than MPALM to ensure δ-presence [6]. The results of the discernibility metric comparison are shown in Figure 6.3. Algorithm 6 has also worse results in all examined cases regarding amount of generalization as shown in 6.4. This experiment demonstrated that algorithm 6 results in bigger information loss and discernibility metric value, than MPALM, the most efficient generalization based algorithm in all examined cases.

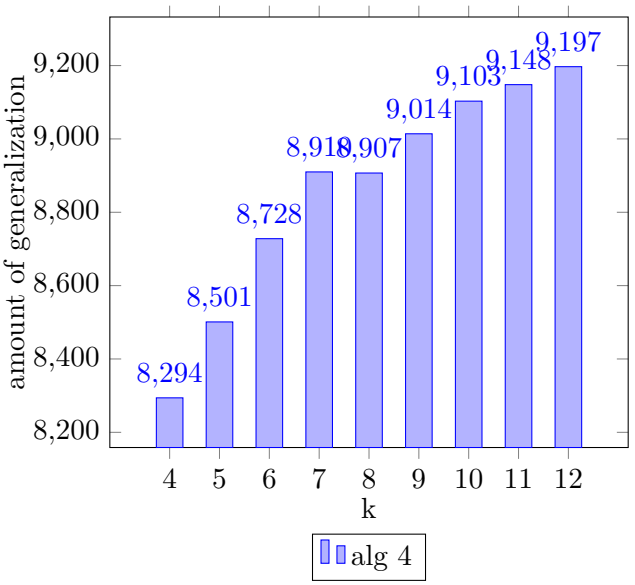

FIGURE 6.5: increasing k in algorithm 6

6.3 Increasing k in algorithm 6

In the third experiment, I examine how the amount of generalization changes, as k is increased in algorithm 6. $\delta_{\min}$ was set to 0 and $\delta_{\max}$ was set to 0.9 in all cases. As k is increased, the amount of generalization also increased, as it is shown in Figure 6.5. Independently of the value of k, MPALM outperformed algorithm 6 in all cases. When I increased k to 64, algorithm 6 didn't terminate within the 25 hours, during which I ran the experiment, and this is not practical. The running time plot can be seen in figure 6.6. The second and third experiments demonstrate that (1) algorithm 6 has worse results regarding amount of generalization in the examined cases, than MPALM, and algorithm 6 might be not usable in some cases due to its time complexity, therefore (2) MPALM, the most efficient multidimensional generalization based algorithm is recommended for anonymization, if a published dataset has to be protected against membership disclosure.

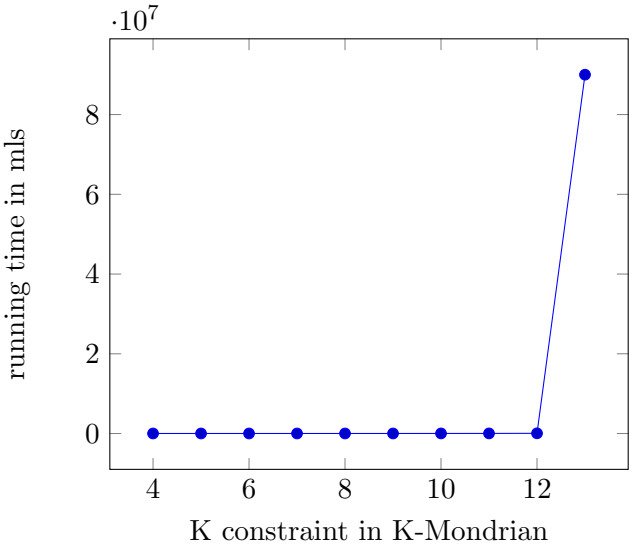

FIGURE 6.6: running time of algorithm 6

Chapter 7

Conclusion and future work

In this thesis, I used a table linkage model which supposes that the adversary has access to an external public table P, and private table $T \subseteq P$. Deciding about which table linkage model is the most relevant is the subject of future work.

The main contribution of this master's thesis is that it shows that permutation based algorithms don't provide a strong protection against membership disclosure. If membership disclosure is crucial, some degree of generalization is unavoidable. Comparing generalization based techniques to differential privacy is also the subject of future work.

I proved that specialization cannot be applied on the result set of MPALM without loosing the δ-presence property, therefore any algorithm combining permutation and generalization will have bigger bucket sizes/ worse results regarding discernibility cost metric, than MPALM.

An algorithm (algorithm 6) was introduced that combines generalization with permutation and enforces δ-presence. In the experiments, I demonstrated that MPALM, the state of art generalization based algorithm, outperforms algorithm 6 in all of the examined cases regarding discernibility metric and amount of generalization. Therefore I can conclude that multidimensional generalization in itself results in less information loss and discernibility metric value, than permutation combined with multidimensional generalization, if protection against membership disclosure is crucial. Algorithm 6 might produce better results, than MPALM regarding amount of generalization in some rare cases, but it is NP-hard, therefore it is not usable in many cases.

The conclusion of this thesis is that if a published dataset has to be protected against membership disclosure, generalization based techniques are recommended for anonymization.

Chapter 8

Acknowledgments

Thanks to my supervisor Carsten Schürmann for the useful comments, remarks and the help with refining the proof in the thesis. Moreover, thanks to Troels Bjerre Sørensen for the help with Algorithm 5.

Appendix A

Notes on the implementation

I write notes about my implementations here, because it doesn't belong to the main text of the thesis. I have attached some code to my thesis. In the attachments, there is a folder called AnonymizationLibrary. It contains the implementations of the algorithms that I used in the experiments. I wrote all of codes in this library. Inside the Visual Studio Project, there is a console application project called ExampleUsage. In this project, there are some examples for running the implementations of the algorithms.

Another side project of this thesis was to design and implement an anonymization tool for a data warehouse company called Rehfeld Partners A/S. I would like to emphasize that this side project doesn't belong to the main part of my thesis. In this project, I have worked together with two other students (Andrea Del Popolo and Simone Leomanni) from Copenhagen University. In the attachments, there is another folder called Data Anonymization Tool Project. It consists of 3 Visual tudio projects and the design specification document of the data anonymization tool. One of the Visual Studio projects contains the data anonymization tool. The data anonymization tool uses principles of plugin architecture. I have only implemented and integrated the K-Mondrian algorithm plugin in the data anonymization tool; the other parts were written by the other students. Another Visual Studio project is called SSISPoc. It contains a SSIS (Microsoft SQL Server Integration Services) project that uses the data anonymization tool. SSIS is a platform for data integration and workflow applications, and it is heavily used in Rehfeld Partners A/S. In the future, the anonymization tool is going to be used mainly in SSIS projects. Moreover, there is another Visual Studio project which uses the data anonymization tool. It is called 'DataAnonymizationToolExampleUsage'. It is a console application project, and it is easier to set it up, than the SSIS project. It also consists of a ReadMe.txt with setup instructions.

Bibliography

[1] Health insurance portability and accountability act. 1996. URL http://www.cms.gov/Regulations-and-Guidance/HIPAA-Administrative-Simplification/HIPAAGenInfo/index.html?redirect=/HIPAAGenInfo/.

[2] European community directive 95/46/ec. 1995. URL http://eur-lex.europa.eu/LexUriServ/LexUriServ.do?uri=CELEX:31995L0046:en:HTML.

[3] K.E. Emam. *Guide to the De-Identification of Personal Health Information*. CRC Press, 2013. ISBN 9781482218800. URL http://books.google.dk/books?id=Bc8nAAAAQBAJ.

[4] Khaled El Emam and Luk Arbuckle. *Anonymizing Health Data: Case Studies and Methods to Get You Started*. O'Reilly Media, Inc., 1st edition, 2013. ISBN 1449363075, 9781449363079.

[5] Benjamin C.M. Fung, Ke Wang, Ada Wai-Chee Fu, and Philip S. Yu. *Introduction to Privacy-Preserving Data Publishing: Concepts and Techniques*. Chapman & Hall/CRC, 1st edition, 2010. ISBN 1420091484, 9781420091489.

[6] Christopher W. Clifton M. Ercan Nergiz, Maurizio Atzori. Hiding the presence of individuals from shared databases. In *in SIGMOD, 2007*, pages 665–676, 2007.

[7] Latanya Sweeney. K-anonymity: A model for protecting privacy. *Int. J. Uncertain. Fuzziness Knowl.-Based Syst.*, 10(5):557–570, October 2002. ISSN 0218-4885. doi: 10.1142/S0218488502001648. URL http://dx.doi.org/10.1142/S0218488502001648.

[8] Ashwin Machanavajjhala, Daniel Kifer, Johannes Gehrke, and Muthuramakrishnan Venkitasubramaniam. L-diversity: Privacy beyond k-anonymity. *ACM Trans. Knowl. Discov. Data*, 1(1), March 2007. ISSN 1556-4681. doi: 10.1145/1217299.1217302. URL http://doi.acm.org/10.1145/1217299.1217302.

[9] Pierangela Samarati and Latanya Sweeney. Protecting privacy when disclosing information: k-anonymity and its enforcement through generalization and suppression. Technical report, 1998.

[10] P. Samarati. Protecting respondents' identities in microdata release. *IEEE Trans. on Knowl. and Data Eng.*, 13(6):1010–1027, November 2001. ISSN 1041-4347. doi: 10.1109/69.971193. URL http://dx.doi.org/10.1109/69.971193.

[11] Kristen LeFevre, David J. DeWitt, and Raghu Ramakrishnan. Incognito: Efficient full-domain k-anonymity. In *Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data*, SIGMOD '05, pages 49–60, New York, NY, USA, 2005. ACM. ISBN 1-59593-060-4. doi: 10.1145/1066157.1066164. URL http://doi.acm.org/10.1145/1066157.1066164.

[12] Maurizio Atzori. Weak k-anonymity: A low-distortion model for protecting privacy. In Sokratis K. Katsikas, Javier Lopez, Michael Backes, Stefanos Gritzalis, and Bart Preneel, editors, *ISC*, volume 4176 of *Lecture Notes in Computer Science*, pages 60–71. Springer, 2006. ISBN 3-540-38341-7. URL http://dblp.uni-trier.de/db/conf/isw/isc2006.html#Atzori06.

[13] Qing Zhang, Nick Koudas, Divesh Srivastava, and Ting Yu. Aggregate query answering on anonymized tables. In Rada Chirkova, Asuman Dogac, M. Tamer Özsu, and Timos K. Sellis, editors, *ICDE*, pages 116–125. IEEE, 2007. URL http://dblp.uni-trier.de/db/conf/icde/icde2007.html#ZhangKSY07.

[14] Kristen LeFevre, David J. DeWitt, and Raghu Ramakrishnan. Mondrian multidimensional k-anonymity. In *Proceedings of the 22Nd International Conference on Data Engineering*, ICDE '06, pages 25–, Washington, DC, USA, 2006. IEEE Computer Society. ISBN 0-7695-2570-9. doi: 10.1109/ICDE.2006.101. URL http://dx.doi.org/10.1109/ICDE.2006.101.

[15] Roberto J. Bayardo and Rakesh Agrawal. Data privacy through optimal k-anonymization. In *Proceedings of the 21st International Conference on Data Engineering*, ICDE '05, pages 217–228, Washington, DC, USA, 2005. IEEE Computer Society. ISBN 0-7695-2285-8. doi: 10.1109/ICDE.2005.42. URL http://dx.doi.org/10.1109/ICDE.2005.42.

[16] Tiancheng Li, Ninghui Li, Jian Zhang, and Ian Molloy. Slicing: A new approach to privacy preserving data publishing. *Proceedings of Security and Privacy*, pages 64–78, 2009. URL http://arxiv.org/pdf/0909.2290v1.pdf.

[17] Xiaokui Xiao and Yufei Tao. Anatomy: Simple and effective privacy preservation. In *Proceedings of the 32Nd International Conference on Very Large Data Bases*, VLDB '06, pages 139–150. VLDB Endowment, 2006. URL http://dl.acm.org/citation.cfm?id=1182635.1164141.

[18] Xianmang He, Yanghua Xiao, Yujia Li, Qing Wang, Wei Wang, and Baile Shi. Permutation anonymization: Improving anatomy for privacy preservation in data publication. In Longbing Cao, Joshua Zhexue Huang, James Bailey, Yun Sing Koh, and Jun Luo, editors, *PAKDD Workshops*, volume 7104 of *Lecture Notes in Computer Science*, pages 111–123. Springer, 2011. ISBN 978-3-642-28319-2. URL http://dblp.uni-trier.de/db/conf/pakdd/pakdd2011-w.html#HeXLWWS11.

[19] Adrian Akison. Conbinatorics. 2008. URL http://www.codeproject.com/Articles/26050/Permutations-Combinations-and-Variations-using-C-G.

[20] C. Blake and C. Merz. Uc irvine machine learning repository. 1998. URL https://archive.ics.uci.edu/ml/datasets/Adult.

[21] Vijay S. Iyengar. Transforming data to satisfy privacy constraints. In *Proceedings of the Eighth ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '02, pages 279–288, New York, NY, USA, 2002. ACM. ISBN 1-58113-567-X. doi: 10.1145/775047.775089. URL http://doi.acm.org/10.1145/775047.775089.