FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Fingerprinting Relational Databases

Quality Evaluation and Impact on Learning Tasks

DIPLOMARBEIT

zur Erlangung des akademischen Grades

Diplom-Ingenieurin

im Rahmen des Studiums

Logic and Computation

eingereicht von

Tanja Šarčević
Matrikelnummer 01640006

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Andreas Rauber
Mitwirkung: Mag.rer.soc.oec. Dipl.-Ing. Rudolf Mayer

Wien, 4. September 2019

Tanja Šarčević	Andreas Rauber

FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Fingerprinting Relational Databases

Quality Evaluation and Impact on Learning Tasks

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

Diplom-Ingenieurin

in

Logic and Computation

by

Tanja Šarčević
Registration Number 01640006

to the Faculty of Informatics

at the TU Wien

Advisor: Ao.Univ.Prof. Dipl.-Ing. Dr.techn. Andreas Rauber
Assistance: Mag.rer.soc.oec. Dipl.-Ing. Rudolf Mayer

Vienna, 4th September, 2019

Tanja Šarčević | Andreas Rauber

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.ac.at

Erklärung zur Verfassung der Arbeit

Tanja Šarčević
Schäffergasse 2/418, 1040 Vienna, Austria

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 4. September 2019

Tanja Šarčević

Acknowledgements

I thank my parents for giving me unconditional and unlimited support, love and understanding through the years of my education. I thank them for teaching me the importance of hard work and constant learning. Without them and their set of values, which they have been so tirelessly passing on to me, this thesis would not be possible.

I thank my brother, my grandparents and the rest of my family for their patience, support and valuable advice.

I express my sincere gratitude to my advisor, Rudolf Mayer, for his time, guidance, cooperation and patience during the process of writing this thesis. His encouragement helped me, not only during the period of writing the thesis but in academic progress in general, as well. Many thanks to my supervisor, prof. Andreas Rauber, for his help and support.

I would like to acknowledge all of my friends that shared with me the journey from the very beginnings of my master studies until this moment. I am grateful for keeping strong connections with my friends from the homeland, Croatia and for the new friends that I made during my studies in Vienna.

Abstract

Fingerprinting digital data is a method of embedding a traceable mark into the data to verify the owner and identify the specific recipient a certain copy of data set has been released to. This is crucial for releasing data sets to third parties, especially if the release involves a fee, or if the data contains sensitive information due to which further sharing and potential subsequent leaks should be discouraged and deterred from. Fingerprints generally involve distorting the data set to a certain degree, in a trade-off to preserve the utility of the data versus the robustness and traceability of the fingerprint. Different types of data require different approaches. Most of the state-of-art techniques are designed specifically for the numerical type of data. In this thesis, we will propose an approach for fingerprinting data sets containing categorical data. We further compare several approaches for fingerprinting according to their robustness against various types of attacks, such as subset or bit-flipping attacks, and evaluate the effects the fingerprinting has on the utility of the datasets, specifically for Machine Learning tasks.

Contents

CHAPTER 1

Introduction

1.1 Motivation

Over the last decades, the trends of sharing and processing digital data have vastly increased. Data is a valuable asset to its owner, therefore any type of unauthorised distribution or usage by the third parties violates the rights of the owner and authorised buyers. It is the interest of the data owner to detect data leakages and to prove the ownership of the data.

When the owner wants to share the digital data and at the same time keep her rights to it, as an attempt to prevent the unintended sharing or use in the data, one can think of a piece of information to be added to this data so that, when seeing a copy of the data, it is possible to extract the mark that verifies the owner. This piece of information is called a watermark. One type of watermarks are the visible ones, commonly seen in the domain of pictures, videos or PDF files as a visual mark overlaying a part of the data. These watermarks are easy to spot and, with a little bit of skill and effort, removed. However, watermarks can be constructed such that it is nearly impossible for the human observer to see them. Consider choosing a few pixels in an image and slightly changing their intensities. The naked eye will hardly notice any changes, however, the owner can still extract the mark. The model that defines the watermark embeds it in the data and extracts it from the data is called the watermarking technique. Such a technique provides ownership protection for the distributed data.

One step further in ownership protection is detecting the source of unauthorised leakage, i.e. the party that distributed the data without the owner's authorisation. Fingerprinting is another class of steganography, i.e. information hiding techniques, which strengthens the ownership protection by adding the property of leakage source detection and therefore is usually in the literature considered as an extension to watermarking. By fingerprinting

a small piece of information that represents a buyer's identification is incorporated into the data, producing a distinct copy of the data for each of the data receivers. This buyer-specific piece of information is called fingerprint. This area has been widely studied in the domain of multimedia data, while in the area of relational databases there are a rather few proposed approaches.

Usually, the fingerprint is embedded in the database such that the values are slightly altered, which can affect the quality of the data. Once embedded in the data, the fingerprint should be detectable only by the data owner, and it should not be easily removed from the data by any operations on the database, such as removing or adding tuples. Altering the data might sound like a limitation for the buyer, but according to the majority of the proposed techniques, both for watermarking and fingerprinting, this is mandatory to achieve the watermark/fingerprint robustness. Therefore, one has to start with the assumption that for the potential buyers and users of the data these small alterations introduced by watermark or fingerprint are acceptable. The challenge is to find a good trade-off between robustness and data quality and utility.

The technique of fingerprinting is used to protect digital data. Digital data inside of a file can be compared, shuffled, deleted, modified, etc., therefore one must address the problems of different attacks on the fingerprint detection process. The attack may be originating from a malicious attacker who wants to, for example, remove the fingerprint from the data and distribute it illegally without consequences, or wants to falsely claim the ownership of the data, or confuse the fingerprint detection such that some other innocent buyer is accused.

All of the proposed fingerprinting techniques generally imply changes in the values. The errors are usually minor, however, in some settings, this might cause significant violations of the result accuracy made by observation of the data. For example, using a fingerprinted dataset in data mining and knowledge extraction might affect the learning process and the predictions. Since Machine Learning plays a big role in technical progress of today's data-driven economies, it is important to show that these changes in data do not affect the insights inferred from data or a prediction-making process in a way that they significantly reduce the performance of the data mining algorithms.

The alterations have a significant impact on categorical values, which makes fingerprinting categorical data a separate case study. In these settings, it is hard to speak about "minor" error or a scalable measurement for information loss, i.e. the changes are more perceptible. Area of fingerprinting categorical data is not as well studied as fingerprinting numerical data because of the mentioned limitations, even though fingerprinting categorical data is no less important than fingerprinting numerical values.

The fingerprint scheme should satisfy the following properties:

Detectability The owner should be able to detect and distinguish a fingerprint from the dataset. The fingerprint should be detectable also from a subset of the data, as well as the modified version of the dataset.

Imperceptibility The utility of the data should not be significantly affected by modifications caused by fingerprinting. In the literature, this is usually measured by changes in mean and variance of the numerical attributes. Besides mean and variance, we measure data imperceptibility by measuring the performance of the fingerprinted data on a classification task using different classifiers and comparing it to the performance of classifiers trained using unmarked data.

Robustness Fingerprinting schemes should be robust against benign operations on a dataset and malicious attacks that may remove or modify embedded fingerprint. Benign database operations are those with no aim of unauthorised usage or release of the database, such as deleting, adding and updating tuples. Malicious attacks include selective modifications of the fingerprinted database, releasing a subset of a database or modifying and erasing the embedded fingerprint. The malicious attacks that we consider in the thesis, as commonly mentioned in the literature, are:

1. Subset attack: the attacker releases only a subset of the fingerprinted dataset
2. Superset attack: the attacker adds additional tuples to the fingerprinted dataset
3. Bit-flipping attack: values of some bits in the fingerprinted data are inverted so that fingerprint cannot be detected correctly
4. Additive attack: a special case of bit-flipping attack where the attacker adds a fingerprint on the fingerprinted data that might distort the initial fingerprint
5. Collusion attack: attackers with access to multiple copies of fingerprinted data (with different fingerprints) create a new copy of the data where neither of the embedded fingerprints might be detectable, thus no member of the malicious coalition might be implied as a source of leakage

The evaluation also provides insight into relations between different robustness, imperceptibility and detectability levels.

1.2 Problem Statement and Research Questions

Embedding a mark in data is achieved by changing its values. While the fingerprinting technique must assure visibility of a fingerprint to the data owner and imperceptibility to all the other users, it is also important that utility of the data is preserved as much as possible, otherwise, data loses its value. Loss in utility can be analysed in the aspect of changes within the dataset and from the data application point of view, such as training Machine Learning models. Fingerprinting scheme should contain evaluation on both robustness of a fingerprint and utility of fingerprinted data.

The type of data in the dataset can also be a crucial aspect of evaluating fingerprinting scheme effectiveness. Categorical data are shown to give rise to more problems with embedding the fingerprint compared to numerical data, yet an appropriate fingerprinting scheme for categorical data is necessary, otherwise, the domain of fingerprinting applications is very limited.

This work thus considers the following research questions:

1. Which metrics can be recommended to evaluate the robustness of the fingerprinting schemes for relational datasets and which to evaluate the utility of data?
2. How robust are the fingerprinting techniques in the setting of certain malicious attacks?
3. How does the fingerprint in the data affect the quality and the utility of data?
4. How can we design a robust fingerprinting scheme that can be applied on non-numerical data?
5. How can we provide guidance on the application of the most suitable fingerprinting techniques and parameters that meet both the ownership protection and utility requirements for a given data-analysis setting?

1.3 Aim of the work

This work aims to answer the research questions from Section 1.2. Different existing fingerprinting methods will be analysed and their robustness under different attacks will be evaluated, both theoretically and experimentally. Special attention will be given to defining and analysing a method for fingerprinting categorical data. Furthermore, the utility of fingerprinted data will be evaluated concerning changes in metrics such as mean and variance, and concerning effects on Machine Learning models when trained on fingerprinted data compared to models trained using original data. The work will summarise all of the aspects of the fingerprinting scheme, from detectability by the owner, imperceptibility by the users, resilience to attacks and quality and utility of fingerprinted data.

1.4 Methodological Approach

Implementation of fingerprinting techniques and framework for robustness analysis The proposed fingerprinting techniques [1, 2, 3] provide well-defined schemes and robustness analysis but lack in implementations. To perform the empirical evaluation, the approaches are implemented as a part of this thesis. The implementation follows the proposed algorithmic steps from their respective fingerprinting methods and allows to easily change the parameter setting for the fingerprinting method. Secondly, we design and implement a method for fingerprinting categorical data. Finally, robustness analysis

requires a suitable framework for evaluating fingerprinting techniques' resilience against different attacks. Each attack is modelled and incorporated into the framework such that it can be tested on any of the implemented fingerprinting schemes.

Evaluation of the quality effects on a fingerprinted dataset for different fingerprinting techniques As mentioned before, the fingerprint brings a certain distortion to the data. Therefore one should address the problem of modifications of the values and consistency and integrity of the fingerprinted dataset. In this part, we will obtain some measures such as mean and variance of distinct numerical attributes in the real data, analyse those results and compare them to theoretical results, discussed in the respective original papers of the fingerprinting methods [1, 2, 4].

Evaluation of fingerprint robustness for different fingerprinting techniques The second part of this thesis is an evaluation of the robustness of each fingerprinting method against the malicious attacks and benign operations on the dataset. Some of the attacks include bit-flipping attack, subset attack, collusion, etc.[5]. Every fingerprinting method is well substantiated by the probabilistic model for the robustness against each of the attacks. We further run experiments on real data to measure the success of each attack empirically and deliver conclusions and comparisons to the theoretical results. The experiments are run with different parameter settings, therefore the impact of each parameter on the techniques' resilience to attacks is elaborated.

Empirical evaluation of the impact of a fingerprinted dataset on a specific learning target The goal of the last part of the thesis is to evaluate the impact of a fingerprint embedded into the dataset used for a specific learning target. We refer to this impact as a quality effect. The biggest focus is given to measuring the difference in the utility of a dataset before and after fingerprinting under the same parameter setting. This is achieved by performing the same classification tasks on both datasets, in several different experiments with different supervised Machine Learning tasks, different classifiers, i.e. logistic regression, random forest, etc. and different parameters.

The overall goal of the thesis is to deliver a comprehensive overview and analysis of the state-of-the-art fingerprinting techniques for relational databases, present the robustness and defence models against numerous attacks, analyse the distortions to integrity and quality effects of the dataset via impact on learning tasks, corroborate the analysis with the experimental results and provide guidance for choosing a technique and the parameter setting based on a thorough experimental evaluation.

1.5 Structure of the work

Chapter 1 provides the introduction of the thesis. It includes the motivation, problem statement, research questions and aim of the work. Chapter 2 provides the state of the art. It covers the general watermarking and fingerprinting techniques developed for

domains of multimedia, text, software, etc., and related work in the area of watermarking and fingerprinting relational datasets. Chapter 3 describes the chosen fingerprinting techniques. Chapter 4 covers the robustness analysis of chosen fingerprinting techniques against different attacks and empirical evaluation using different datasets. In Chapter 5 we analyse the quality effects of fingerprinted datasets through the experiments on real data and evaluate the effects of an embedded fingerprint in data used for training Machine Learning models. We summarise by providing the recommendations and guidance on the choice of fingerprinting techniques and parameters for specific settings.

CHAPTER 2

State-of-the-Art

Digital watermarking is a method that helps protect intellectual property for various types of digital data. Since it just marks data but doesn't control access to the data, watermarking is a passive protection tool. There is a wide range of applications that watermarking can be used for, such as copyright protection, fraud and tamper detection, content management on social networks, etc. *Fingerprinting* is used for source tracking. It is the special application of watermarking where different recipients of the data get differently watermarked content. The first watermarking methods were developed for the multimedia domain (images, audio, video) and later extended to other types of digital data such as text, software, relational databases, etc.

2.1 Generic framework for watermarking and fingerprinting

Digital watermarking consists of two main processes: *insertion (embedding)* and *detection (extraction)*. The insertion process is in charge of watermark creation and embedding it into the data. The output is the marked copy of the data that is publicly distributed. Insertion process of fingerprinting embeds the different mark for each distributed copy that is specific for each buyer. Detection process extracts the watermark from the data. It reports the existence of a watermark in the data given as the input, identifies the owner, or in case of fingerprinting identifies the buyer. The data on the input of the detection process can be affected by some benign transformations of the data or malicious transformations made by the attacker. The general framework for watermarking (and fingerprinting) is shown in Section 2.1.

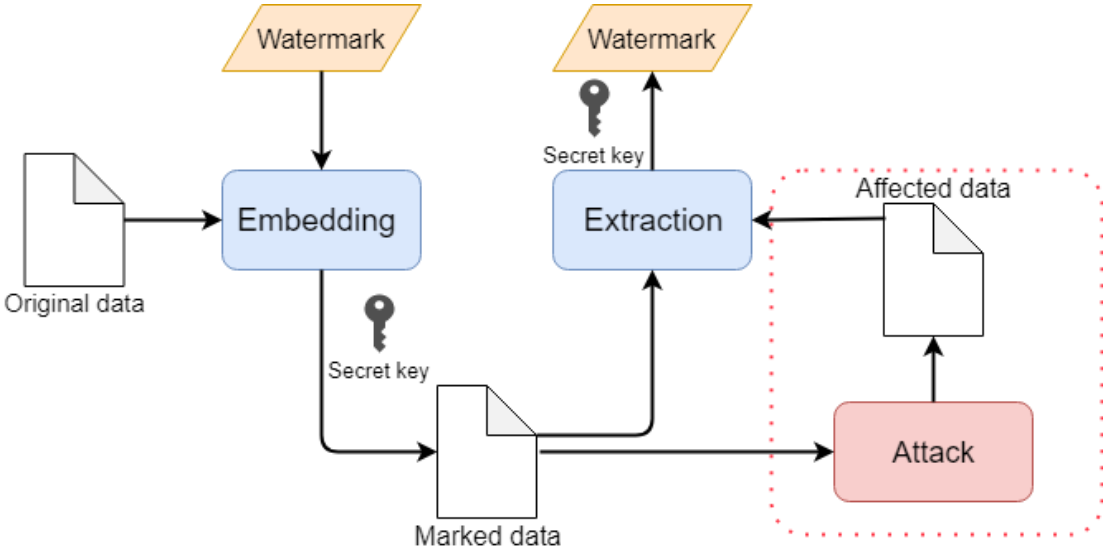


Figure 2.1: General framework for watermarking

2.2 Watermark categorisation

Digital watermarks may be categorised in several ways depending on different properties. The following categorisation criteria are applied to fingerprints as well since we consider fingerprinting as a special case of watermarking.

Imperceptibility The most general categorisation is according to the imperceptibility of a watermark. A digital watermark is called *imperceptible* if the marked copy is perceptually indistinguishable from the original data. A watermark is *perceptible* if its presence in the data is noticeable (e.g. owner's logo as an overlay on an image).

Detectability Watermark schemes can be categorised in terms of verifiability / detectability, i.e. needing the original data to extract the watermark from the marked data. A *Blind* watermarking scheme does not need the original data for the extraction process, while *non-blind* does.

Robustness Furthermore, according to its robustness, watermarks can be categorised into *fragile*, *semi-fragile* and *robust*. Fragile watermarks are commonly used for tamper detection because they fail to be detected after the slightest modification. Semi-fragile transformation resists some benign transformations, but fail detection after malignant transformations (attacks). Robust watermarks resist a wide range of benign or malignant transformations and satisfy the property that the removal of a watermark is not possible without significantly degenerating the data quality.

Distortion In *distortion-based* watermarking techniques marking introduces distortions to the underlying original data. On the contrary, *distortion-free* watermarking techniques rely on detecting the watermark as a function of the data itself without changing the original data. Distortion-free watermarks are fragile and are applied in tamper detection.

2.3 Multimedia, text and software

The concepts of watermarking [6, 7, 8] and fingerprinting [9] digital data firstly appear in domains of multimedia data and are extensively studied over the last two decades. Most of these techniques are developed for images [10] and later extended to video [11] and audio [12, 13].

Image watermarking is applied in two main domains of an image: in the spatial domain where an image is represented by pixels and in the transform domain where it is represented in terms of its frequencies (the image is segmented into multiple frequency bands using a mathematical transform, for instance, Discrete Cosine Transform (DCT), Discrete Wavelet Transform (DWT) and Discrete Fourier Transform (DFT)) [14]. The watermark can be embedded in the spatial domain by modifying pixel values [15] or adding an overlying layer on top of the original image [16], however more robust watermarks are usually embedded in the transform domain by modifying the transform domain coefficients [17, 18]. More advanced watermarking techniques are developed for video watermarking which extends and incorporates watermarking techniques for images [19, 20]. Most of the image and video watermarking algorithms focus on processing data files stored on a disc, while some introduce real-time content processing (applied for example on screenshot images) [16, 21].

There have also been some approaches in applying a watermarking scheme in other domains such as text and software. Techniques for watermarking text data typically exploit special properties of text formatting and semantics. Watermarks are often introduced by altering the spacing between words and lines of text [22]. Other techniques rely on natural language processing and rephrasing sentences in the text [23, 24, 25, 26]. Techniques for watermarking software only have limited success in their native domain [27, 28]. A key problem is that the instructions in a computer program can often be rearranged without altering the semantics of the program. This resequencing can destroy a watermark. Techniques have also been proposed to prevent copying of software, but they require installation of tamper-resistant modules in users' machines, which limits their successful adoption in practice.

2.4 Watermarking Relational Databases

Digital watermarks in the domain of multimedia are not easily extended or adapted to relational database applications, because of the inherent differences between multimedia data and relational databases. For example, a lot of techniques for images and audio rely on phenomena based on the limitation of the human visual and auditory system.

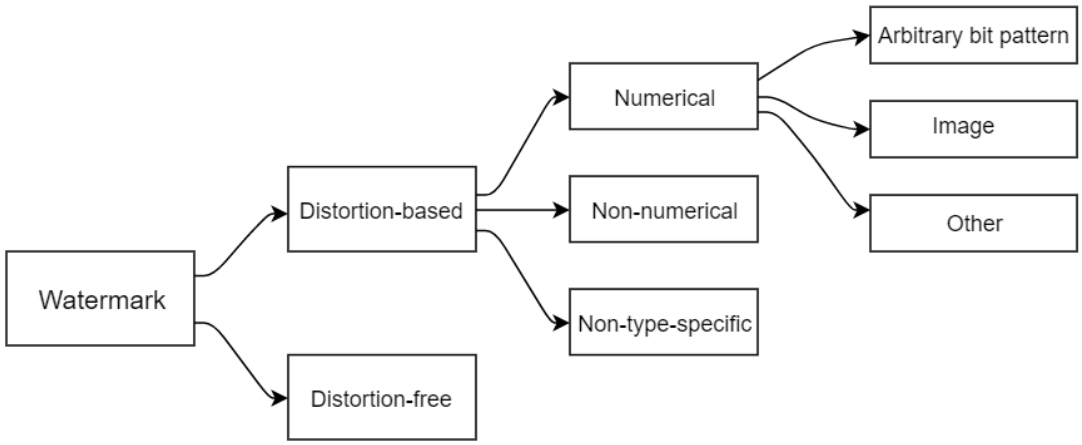


Figure 2.2: Categorization of watermarking techniques for relational data

Furthermore, multimedia data contains a large number of redundant bits providing much wider cover for embedding the watermark than it is the case with relational data. Relational data may as well consist of multiple types of data - numerical, categorical, text, etc., requiring different approaches for embedding the watermark. Categorical data is known to introduce additional issues for watermarking compared to numerical, so we will divide these techniques into separate subsections. Therefore, techniques for watermarking relational data differ significantly from those for multimedia. A couple of surveys [29, 30, 31, 32] list most of the proposed watermarking techniques in the literature and classify them, comparing thereat their main properties and robustness to attacks. In the remainder of the section, we present the watermarking techniques for relational data, according to the categorisation illustrated in Figure 2.2 [5]. The first level of categorisation is a distinction of watermarks based on distortion. Distortion-based watermarks are then divided into categories based on the type of data they apply to. A further distinction is made among the techniques applied to a numerical type of data according to what carries the watermark information.

2.4.1 Distortion-based watermarks

Watermarking numerical data

1. *Arbitrary bit pattern as watermark information*

   Pioneering work on watermarking relational databases is proposed by Agrawal and Kiernan [33]. It is a blind, bit-resetting technique where the watermark insertion is controlled by a gap parameter γ (the ratio of tuples to be fingerprinted). They use a message authentication code (MAC) - a one-way hash function that depends on a secret key to determine which tuples will be marked. The same value $\mathcal{F}(r.P) = \mathcal{H}(\mathcal{K}||\mathcal{H}(\mathcal{K}||r.P))$ is used for selecting the tuple, attribute and bit that will be marked,

where $\mathcal{H}$ is a hash-function, $\mathcal{K}$ is a secret key, $r.P$ the primary key of the tuple and $||$ is representing a concatenation. The important property of a hash function is that it is easy to calculate a hash value (output) for a given input, but computationally difficult to do the reverse. Therefore only the owner knowing the secret key can detect where the marks are embedded in the data.

This work is extended in [34, 35] by introducing a pseudo-random sequence generator instead of the MAC in the process to select tuples, attributes and bits for watermarking. The technique from [35] (referred to as the AKH technique; AKH stands for the initials of the authors) and its predecessor [33] is used for many extended works in watermarking and fingerprinting relational databases. The technique in principle contains two steps: watermark insertion and watermark detection, according to the generic framework for watermarking.

The *watermark insertion algorithm* marks certain numerical attributes such that the least significant bits (LSBs) are altered, therefore this technique assumes that the dataset contains one or more numerical attributes. The database owner is left to decide the number of LSBs available for marking ξ, such that the changes of marked attributes stay imperceptible. The insertion algorithm then uses a cryptographic pseudo-random sequence generator $\mathcal{G}$, seeded by a secret key known only to the owner of the database, and concatenated with the primary key attribute value of each tuple from a database. The main assumption that the AKH technique relies on is the existence of a primary key in the dataset. Li et. al. [36] propose three different schemes to obtain a virtual primary key for relational databases without primary key. The numbers generated by $\mathcal{G}$ determine the tuples, attributes within the tuples and LSBs within the attribute values to be marked, as well as the mark itself. It is computationally infeasible to predict the next number in the cryptographic pseudo-random sequence. Thus, it is computationally infeasible to guess the marking pattern without the knowledge of the owner's private key.

The *detection algorithm* contains calculating the sequence using a cryptographic pseudo-random sequence generator with the same seeds as in the insertion algorithm. Those sequences will be the same because repeated executions of the generator seeded by the same value always produce the same sequence. Thus, the detection algorithm finds which bits within the database should have been marked and counts how many of them match the bits from the database suspected for piracy. If the number of matches is "large", the database owner can suspect the piracy. On the other hand, if the matching is "too small", the database owner can suspect that the attacker somehow identified the marking pattern and recreated the original database values. The number of matches necessary for suspecting piracy is defined by a parameter τ called *significance level*.

The authors analysed the robustness of this technique against several malicious attacks, namely, subset attacks, bit-flipping attacks, mix-and-match attack and false claim of

ownership. More of the robustness analysis of the AKH scheme is done in later work by Lafaye [37].

Extensions and improvements to the AKH technique have been proposed in [38] and [39]. The first one [38] improves the AKH by using chaotic random series based on the Logistic chaos equation instead of a hash function. This has two advantages: non-repetitive iterative operation and sensitiveness to initial values in a way that the selection of bits meets the requirements of both data range and data precision of each attribute, rather than using same ξ for all attributes as it is the case in the AKH. This strategy's advantages manifest by a significantly decreased error when data with large differences in ranges within the attributes is watermarked. Another extension is a reversible version of the AKH watermarking scheme [39] where watermark can be recovered during the detection phase and the attribute can be restored to its unmarked value.

The watermarking method in [40] embeds random digits at second LSB positions of the numeric candidate attributes. The watermark is not explicitly embedded like in the previously mentioned techniques, but rather used for identifying valued groups of tuples to embed the random values.

2. *Image as watermark information*

   There is a couple of proposed techniques that mark the data using an image as watermark information.

   The method in [41] describes embedding the scrambled image based on the Arnold transform with scrambling number d. The scrambling number improves the security of this technique because of it an additional parameter, besides the secret key, that the embedding scheme relies on. The scrambled image is represented as a binary string and embedded into tuples that are previously grouped using hash values depending on a secret key, primary key and the order of the image.

   Another method [42] follows the similar algorithmic steps as in [41], but embeds the original image converted into a bit flow instead of a scrambled image.

   The method in [43] divides an image into *header* and *image data* in the insertion phase. The header is used in a hash function together with tuple's primary key to determine tuple's ID value and determine positions where the image data will be embedded.

3. *Other types of watermark information*

   Besides arbitrary meaningless bit pattern and image as watermark information, other types of watermark information have been proposed and embedded into the data. One example is speech as a watermark information [44]. The proposed technique uses compressed owner's speech converted into a bitstream as a watermark and follows similar algorithmic steps as [42]. Furthermore, in [45] authors propose a Genetic Algorithm-based technique to generate watermark signal, in [46] the cloud

watermarking scheme is proposed and authors in [47] use *k-means* algorithm to cluster the tuples into some equivalent classes.

Watermark information may as well be the content of data itself [48, 49]. Generally, some bits of one part of the data, i.e. characteristics are extracted in the insertion phase and used to mark the other parts of the data.

Watermarking non-numerical data

Non-numerical data requires different techniques for watermarking purposes than numerical data because of its discrete nature. The concept of introduced errors is perceived differently as well because we cannot apply the same measurements as in the case of categorical data, e.g. marking the numerical value 23 as 25 does not have the same impact on the data as marking "blue" as "red". The latter may not even be acceptable in some settings, therefore watermarking non-numerical data needs to start from the assumption that the user accepts this kind of errors and that they do not violate credibility and utility of the data. Another limitation of watermarking categorical data that can cause the loss of credibility is the semantic correlation between the attributes that depending on the setting needs to be preserved.

In [50, 51] the authors propose a technique for watermarking categorical data. The technique requires the presence of a primary key in the dataset. The embedding process relies on two secret keys k_1 and k_2. The first secret key k_1 is used for selecting the tuples "fit" for watermarking together with parameter e and the primary key. e is a control parameter of how many tuples get marked, i.e. has the same purpose as parameter γ in schemes for watermarking numerical data. The number of tuples "fit" to be marked (η/e) is usually larger than the size of a watermark, which is solved by converting the watermark wm into wm_data of length η/e using an Error Correcting Code (ECC). In every "fit" tuple, the embedding algorithm generates a secret value in bit representation. The number of bits in a secret value is equal to the number of bits required to represent all possible categorical values for the attribute. The least significant bit of the value is marked by a randomly chosen bit of wm_data depending on the primary key and the secret key k_2. The marking scheme assures that the resulting categorical value will be some value from the domain of the attribute. The pseudo-random nature of the choice of a bit from wm_data guaranties that almost all bits will be chosen at least once during the embedding process. Using two different secret keys k_1 and k_2 assures non-correlation between the selected tuples for embedding and the corresponding bit value positions in wm_data.

The detection algorithm reverses the operations from the insertion phase and extracts some wm_data'. Once wm_data' is available, the ECC is invoked to generate the "closest" or "most likely" corresponding watermark wm.

The authors also suggest an embedding scheme based on multiple categorical attributes by considering not only the association between the primary key and a single categorical attribute but also associations between different categorical attributes. The embedding

scheme described previously is applied for every combination of *primary key - categorical attribute*, and every combination of two categorical attributes. In combinations of two categorical attributes, one of them serves as a virtual primary key for the embedding process. For instance, let us have a dataset containing primary key K and two categorical attributes C_1 and C_2. The embedding scheme will be performed 3 times, for (K, C_1), (K, C_2) and (C_1, C_2). First, two runs of the embedding scheme will use the secret key k_1, e and the primary key K for the choice of tuples to be marked and embeds the mark in attribute C_1 (or C_2 for the second combination). The embedding scheme run for (C_1, C_2) uses the secret key k_1, e and attribute C_1 for the choice of tuples to be marked and embed the mark in attribute C_2. In this scheme for watermarking multiple categorical attributes, taking just described case, we can see that markings for (K, C_2) interfere with markings for (C_1, C_2). Even though this scheme is claimed to be robust against serious attacks (e.g. vertical data partitions) and breaks the dependency of the primary key, the convenience of this method is questionable due to the increase of the number of combinations (sequence of triangular numbers) for more categorical attributes in the dataset.

This watermarking technique is applied to binned medical data in [52].

Another method for watermarking non-numerical data has been proposed in [53], for non-numeric multi-word attributes. The watermarking scheme is based on hiding a binary image in space of non-numeric multi-word attributes of subsets of tuples by introducing double spaces on pseudo-randomly chosen places. The scheme is claimed to be robust against subset and superset attacks, however, it may suffer from the simple malicious action that replaces all double spaces between two words by a single space and therefore erases the watermark.

Watermarking non-type-specific data

Techniques have been proposed to use fake tuples [54] or fake attributes [55] as watermark information.

The approach from [54] generates fake tuples and inserts them erroneously into the dataset. The fake tuple creation algorithm uses Bernoulli sampling probability to decide whether the new value will be chosen from the existing set of values of the corresponding attribute, or a set of fake values. The choice of the new value can be made uniformly, or as the value with higher-occurrence frequency in the existing set of values of the corresponding attribute in the relation. The detection algorithm checks via the primary key to see whether the fake tuples inserted during watermarking insertion phase exist or has been changed. As soon as it finds one match, detection is done. The detection fails for the watermarked database when all of the fake tuples are deleted. One advantage of this scheme is that the owner can be publicly verified more than once until all of the fake tuples are revealed. Another advantage is that this scheme works with all types of values - numerical, categorical, dates, etc.

The technique of inserting a virtual attribute in the relation which serves as watermark [55] generates new values as aggregates obtained from any other numerical value from the dataset. This approach is fragile and can easily detect any of the deletion, insertion or alteration attacks, however, it suffers from the watermark removal attack.

2.4.2 Distortion-free Watermarking

The watermark information may be the hash value extracted from the data. The main idea is to partition the data in some way. [56] propose a technique where partitioning of tuples is based on the hash value parameterized with the primary key and secret key, whereas in [57], partitioning is based on categorical attribute values. After partitioning, hash values for each group as well as tuple level hash values are computed. Based on the hash values and their parity, two tuples' order changes or doesn't. These schemes are able to detect any modifications made to a database relation.

There are other proposed solutions in the literature that are used as watermark information, such as combining owner's mark and database features [58], converting database relation into binary form [59, 60] or R-tree based permutations [61].

Most of the distortion-free watermarking techniques are fragile, i.e. they aim at maintaining the integrity of the information in the database in addition to the ownership protection. The watermark insertion phase does not depend on any specific type of data and does not introduce any distortion in the underlying data of the database.

2.5 Fingerprinting Relational Databases

Fingerprinting is a special application of watermarking - a distinct watermark (a fingerprint) is generated and embedded in every distributed copy of the data. In paper [32], the authors gave a classification and brief analysis of relevant watermarking and fingerprinting techniques. Table 2.1 is derived from this survey. In our review we mention many of the techniques from [32] and include some that are not mentioned in the survey.

Li et al. in [1] extended the watermarking technique proposed by Agrawal et al. [35] into a fingerprinting technique. The marking scheme embeds different bit-strings - *fingerprints* in different releases of data. The owner generates buyer's fingerprint from the owner's secret key and the buyer's serial number using a cryptographic hash function. This method of generating the fingerprints avoids storing buyer-fingerprint pairs and additional security management for this database. Similarly to the watermarking technique [35], this fingerprinting technique consists of the insertion algorithm and detection algorithm.

The insertion algorithm follows the steps for selecting the tuple, attribute, LSBs and marks in the same manner as the insertion scheme from [35], and additionally embeds the generated fingerprint by XOR function applied on the mark (in this algorithm called *mask*) and a selected fingerprint bit.

The aim of the detection algorithm of the fingerprinting methods, in general, is to determine whether the suspicious database is pirated, as well as to identify the source of the unauthorised release of the database. The detection algorithm from [1] reverts the fingerprint insertion process similarly to [35]. It locates the bits that should have been altered and compares the matching of the extracted fingerprint with buyers' fingerprints. The exact bit matching of the extracted fingerprint to some buyer's fingerprint implies this buyer to be an unauthorised distributor. τ is a parameter related to the assurance of the detection process.

In [2], authors propose a block-oriented fingerprinting scheme for relational databases inspired by a fingerprinting scheme for images from [62]. This method also relies on altering the LSBs at certain locations in the database.

In the insertion algorithm, the LSBs of numerical values from the database (as much LSBs as it is allowed to change) are combined into a two-dimension image and separated into blocks of size $\beta \times \beta$. Then a pseudo-random number generator is used to decide the block and position within the block where the fingerprint should be embedded until all blocks are marked. Fingerprint bits will be embedded again if there is still unmarked blocks left when all fingerprint bits have been embedded. A fingerprint is produced in the same manner as in previous techniques, using the cryptographic hash function seeded by the owner's secret key and the user's serial number.

Detection phase consists of sorting the suspicious database according to the primary keys and filling out the database with the original values in case of deletion. The location where the fingerprint bit is supposed to be is calculated as in insertion algorithm and the bit is recorded. As the fingerprint is embedded multiple times in the dataset, if most of the detected values for a single fingerprint bit are 1, the detected fingerprint is said to be 1, otherwise 0.

Authors of watermarking and fingerprinting system *Watermill* [4, 63] further extend the methods from [35] and [1] by considering the constraints of data alteration and treating fingerprinting as an optimisation problem. By using a declarative language, the usability constraints that the fingerprinted dataset must meet are specified. For example, a purchaser may require that joins between some attributes should be preserved, or that values of some attribute cannot be altered more than a predefined amount. One of two proposed fingerprinting strategies consists of translating the weight-independent constraints into an integer linear program (ILP) and using the ILP solver to solve it. The second fingerprinting strategy is *pairing heuristics* for larger datasets where using ILP solver might not be efficient.

In [3] and in its predecessor [64], the proposed technique is a two-level fingerprinting scheme. In the first embedding process, a distinct fingerprint of length L for each buyer is embedded. The tuples are partitioned into L subsets and in each of them, one fingerprint bit is embedded in a pseudo-random position. The second embedding process is designed

for verifying the extracted fingerprint and numerical confidence level. The selected positions are marked as "0" or "1" depending on the hash value seeded with secret key concatenated with the primary key. A fingerprint from the first level is used as a secret key for the second embedding level. To avoid conflict between two levels of embedding, the bit values in the second embedding process are only considered for marking if not marked in the first embedding process.

In [65] an architecture for identifying a malicious buyer and redistributing digital content is proposed. When a user accesses the database, a fingerprinting process is invoked. A parameter manager module is used to store the fingerprinting parameters and a manager module is in charge for the fingerprinting task and verifying authenticity of users; if the user is unauthentic (not the owner), the fingerprinting process is executed. The encoding and decoding algorithm is the same as in [64].

All of the above fingerprinting techniques have one restriction in common - they are applicable only on numerical attributes. Only a few solutions have been proposed for the categorical data.

One approach is a fingerprinting technique that incorporates the k-anonymity property into the fingerprinted data [66]. k-anonymity [67] strives to modify a dataset so that at least k data samples (individuals) become indiscernible when considering quasi-identifying attributes. This is commonly achieved by generalising values in the dataset to a broader meaning. There are generally multiple solutions for achieving the same level of k by choosing different attributes to modify. The idea of the scheme is, therefore, that equivalent k-anonymous patterns of the given dataset serve as fingerprints, and the process of anonymizing data is at the same time the fingerprinting process. K-anonymity is applied on both categorical data and numerical, therefore this model, unlike the previous, provides the fingerprinting technique that includes the categorical data in the process. However, there are some limitations: (i) the number of available fingerprints is limited to the number of different equivalent patterns for k-anonymity in the dataset, (ii) the utility of the differently fingerprinted (anonymized) datasets can vary significantly, and (iii) the fingerprint cannot be computed alone by the recipient's identifier, but rather, a mapping of fingerprint and recipients needs to be stored, with all associated security risks.

Table 2.1: Summary and comparison of well known fingerprinting techniques

Scheme	Attribute data type	Attribute selection method	Tuple selection method	Granularity level	Detectability	Dependencies
AK [1]	Numerical	PRSG	PRSG	Bit	Blind	Primary key, Value
Block Scheme [2]	Numerical	All	All	Bit	Not Blind	Primary key, Value
Watermill [63]	Numerical	Arbitrary	PRSG	Bit	Blind	Primary key, Value
Two-level scheme [3]	Numerical	Arbitrary	SHF	Bit	Blind	Primary key, Value
FP Architecture [65]	Numerical	Arbitrary	SHF	Bit	Blind	Primary key, Value
k-anonymity based [66]	All	All	All	Value	Blind	Value

CHAPTER 3

Fingerprinting techniques

Fingerprinting is a technique that embeds a piece of information into the data to provide the identification of the owner of data and the source of potential unauthorised data leakage. Fingerprint combines secret owner-specific and buyer-specific information and is embedded in the dataset. Every buyer has her fingerprint, therefore every dataset fingerprinted and distributed by the owner is different from each other. By detecting the fingerprint within the dataset, the owner can trace the buyer of that instance of the dataset. The fingerprinting techniques usually contain two main algorithms: fingerprint insertion and fingerprint detection. In the fingerprint insertion, the fingerprint of a buyer is embedded into the dataset. Fingerprint detection strives for detecting the fingerprint in a suspicious dataset to connect it with the buyer who distributed the dataset without the authorisation. Detection could be disrupted by malicious attempts of the buyer to remove the fingerprint from the data, but also by benign changes in the dataset. These attacks and fingerprint resistance to them will be addressed in Chapter 4.

3.1 Notation and parameters

In the following, several different fingerprinting techniques are discussed in detail. In this section, we set the common notation and explain the parameters used in the techniques. The dataset schema is denoted by $\mathcal{R}(P, A_0, ..., A_{v-1})$ where $\mathcal{R}$ is a database relation, P is the primary key attribute, and $A_0, ..., A_{v-1}$ are v attributes that will be used for fingerprinting. η is a notion of number of tuples (rows/entries) in the dataset. As the fingerprinting techniques embed the fingerprint by changing the attribute values, we define the parameter ξ which denotes the number of least significant bits (LSBs) that can be used to embed the fingerprint.
Let N be the number of buyers to whom the dataset is being distributed. To each of the buyers, a unique fingerprint will be assigned. Every fingerprint $\Gamma = (f_0, ..., f_{L-1})$ is a binary string of length $L \geq logN$.

Table 3.1: The notions of the most common parameters

Notation	Meaning
$\mathcal{R}$	database relation
P	primary key attribute
A_i	attribute i
v	number of attributes
η	number of tuples
$\mathcal{K}$	owner's secret key
ξ	number of least significant bits
$1/\gamma$	ratio of tuples to be marked
N	number of buyers
L	length of fingerprint
ω_i	number of embeddings of a fingerprint bit i
ω	total number of marks

Besides the buyer's fingerprint, the owner-specific information has to be embedded in the dataset to ensure the ownership protection and make it possible only for the data owner to detect the fingerprint from the fingerprinted dataset. This information is usually a secret key $\mathcal{K}$ - a binary string known only to the dataset owner. Table 3.1 contains the notation that will be used in the remainder of this thesis unless otherwise stated.

3.2 Prerequisites

Before discussing the specific schemes and algorithms used for fingerprinting relational databases, it is necessary to mention auxiliary functions and algorithms required to achieve the goal of robust fingerprints. Cryptographic pseudo-random sequence generator and cryptographic hash function, discussed in the following subsections, both ensure that neither the fingerprint nor any part of fingerprinting scheme get disclosed or recreated by an unauthorised user. This stated, their usage is crucial for the owner's rights protection.

3.2.1 Cryptographic pseudo-random sequence generator

A cryptographically secure pseudorandom number generator (CSPRNG) is an algorithm for creating a sequence of random numbers with properties suitable for use in cryptography. The CSPRNG-generated sequence is not truly random, because it is completely determined by the initial value called *seed*. The requirements of CSPRNG fall into two groups:

- They pass statistical randomness tests.

  A generator passing the *next-bit test* will pass all other polynomial-time statistical tests for randomness (proof by Andrew Yao in 1982 [68]). We say that a sequence of bits passes the next-bit test for at any position i in the sequence, if any attacker

who knows the i first bits (but not the seed) cannot predict the $(i+1)$st with reasonable computational power.

- They resist a serious attack, even when part of their initial or running state becomes available to an attacker.

  If part or all of CSPRNG's state has been revealed (or guessed correctly), it should be impossible to reconstruct the stream of random numbers before the revelation. Additionally, if there is an entropy input while running, it should be unfeasible to use knowledge of the input's state to predict future conditions of the CSPRNG state.

There is several designed practical CSPRNGs including the Yarrow Algorithm [69] incorporated in iOS and macOS, its successor Fortuna [70] used in FreeBSD, CryptGenRandom [71] included in Microsoft's Cryptographic Application Programming Interface, etc.

For the analysis in the thesis, we denote CSPRNG with $\mathcal{S}$, where $\mathcal{S}_i$ denotes the i^{th} random number of the sequence.

3.2.2 Cryptographic hash function

The cryptographic hash function is a deterministic function that takes a string input of any length and returns a fixed-size string value. The returned value is called "hash value". In literature the hash value can be also referred to as "digest", "checksum" or "digital fingerprint" (the word *fingerprint* is here used in the different context than the *fingerprint* used as a topic of this thesis). The hash function has three main properties:

- It is easy to calculate a hash value for any given input string.
- It is computationally difficult to calculate an input that has given a certain hash value.
- It is extremely unlikely that two different inputs, even remotely different, have the same hash value.

While the regular hash functions aim to, most importantly, avoid the collision of hash values for non-malicious input, the above properties are much strongly guaranteed for cryptographic hash functions. The hash value serves as a "signature" for the input provided. Only the person knowing the original input value can easily check the matching hash function. However, knowing only the hash value, one is unable to do the inverse and find out the original input value. If either finding a string that matches a given hash value or finding two different inputs that have the same hash value is computationally feasible, then a cryptographic function is not considered secure from a cryptographic point of view.

Most commonly used cryptographic hash functions are MD5 [72] and SHA-1 [73].

3.3 Fingerprint codes

A fingerprinting scheme assigns a specific fingerprint to each of the N buyers. A fingerprint is a unique binary string of length L known only by the database owner which is used to trace a specific owner. Storing the mapping of a recipient to her fingerprint is additional data that requires additional protection measures to be protected against attacks. To avoid this, the owner uses a cryptographic hash function $\mathcal{H}$ to produce each buyer's fingerprint instead. In the fingerprinting techniques presented in the following sections, each fingerprint $\mathcal{F}$ is produced as a hash value of the concatenation of owner's secret key $\mathcal{K}$ and buyer's identification number id:

$$\mathcal{F}(\mathcal{K}, id) = (f_0, ..., f_{L-1}) = \mathcal{H}(\mathcal{K}|id) \tag{3.1}$$

where buyer's identification number can be publicly accessible. This way only the owner who knows both secret key $\mathcal{K}$ and identification number id can produce the hash value - the fingerprint of a specific buyer.

Collusion resistant codes Fingerprinting schemes are susceptible to collusion attacks where users with multiple copies of the same dataset but different embedded fingerprints work in coalition to create a useful data copy that does not implicate any member of the coalition. Collusion attacks will be thoroughly discussed in Section 4.7, while in this section we cover collusion resistant codes. Collusion resistant fingerprinting codes have been studied extensively in the literature [74, 75, 76, 77, 78]. One of the well-known codes is proposed by Boneh and Shaw [74] (this code is referred to as *BoSh code* in the remainder of the thesis). The effectiveness of the BoSh code relies on the assumption that colluding buyers can detect only the fingerprint bits in which their copies differ, otherwise a fingerprint bit cannot be detected. For example, two buyers with their own fingerprinted datasets can easily compare their datasets and remove or change the values that differ between their copies. Authors of [74] call refer to this as the "Marking Assumption". The main property that codes should satisfy is that users cannot change the state of an undetected bit without rendering the dataset useless. This small amount of information where the buyers' marks agree is used to trace the copies they generate back to either of them. A collusion resistant code is expected to have two properties well-elaborated:

- c-frameproof code, i.e. a coalition of at most c users cannot frame a user not in the coalition even all user fingerprints are known to the users
- c-secure code, i.e. there exists a tracing algorithm that on input x must output a single member of the coalition of at most c users, where the input x is the new mark extracted by the fingerprint extraction algorithm that was generated by the coalition

The usage of the BoSh code is discussed in Section 3.4 as a part of the collusion-resistant version of the scheme.

3.4 AK Scheme

Algorithm 3.1: AK Scheme: Insertion Algorithm

Input: dataset $\mathcal{R}$ with scheme $(P, A_0, ..., A_{v-1})$, buyer n's ID id
Output: fingerprinted dataset $\mathcal{R}'$

1 fingerprint of buyer n: $\mathcal{F}(\mathcal{K}, id) = \mathcal{H}(\mathcal{K}|id)$
2 **foreach** *tuple* $r \in \mathcal{R}$ **do**
3 **if** $(\mathcal{S}_1(\mathcal{K}|r.P)\ mod\ \gamma == 0)$ **then**
4 attribute_index $i = \mathcal{S}_2(\mathcal{K}|r.P)$ mod v
5 bit_index $j = \mathcal{S}_3(\mathcal{K}|r.P)$ mod ξ
6 mask_bit $x = 0$ if $\mathcal{S}_4(\mathcal{K}|r.P)$ is even; $x = 1$ otherwise
7 fingerprint_index $l = \mathcal{S}_5(\mathcal{K}|r.P)$ mod L
8 fingerprint_bit $f = f_l$
9 mark_bit $m = x \oplus f$
10 $LSB(j, r.A_i) = m$
11 **end**
12 **end**
13 **return** $\mathcal{R}'$

In [1] the authors describe a fingerprinting scheme that is an extension of the watermarking scheme from [35]. In this thesis, this fingerprinting scheme will be referred to as AK Scheme whose name is made of last name initials of authors of the underlying watermarking technique, Rakesh Agrawal and Jerry Kiernan.

3.4.1 Algorithms

Insertion Algorithm 3.1 shows the pseudo-code of the insertion algorithm, which is used to embed a fingerprint of a buyer n into the dataset $\mathcal{R}$. Using random numbers generated by a pseudo-random sequence generator $\mathcal{S}$ the algorithm chooses the bit within the dataset's values that will be marked, as well as the bit that will be embedded. The pseudo-random sequence generator is independently seeded for each tuple with a concatenation of the owner's secret key $\mathcal{K}$ and the primary key of each tuple. In the line 3, the algorithm decides whether the current tuple will be marked, based on the generated number $\mathcal{S}_1(\mathcal{K}, r.P)$. In lines 4 and 5, based on $\mathcal{S}_2(\mathcal{K}, r.P)$ and $\mathcal{S}_3(\mathcal{K}, r.P)$, it is decided which one of the attribute's values will be marked, and which least significant bit of the value, respectively. In lines 6-10 the algorithm determines which value to replace this bit with. The next numbers, generated by the sequence generator $\mathcal{S}_4(\mathcal{K}, r.P)$ and $\mathcal{S}_5(\mathcal{K}, r.P)$, decide the mask bit and choose the fingerprint bit, respectively. Finally, the resulting bit that is embedded in the line 10 at the chosen place in a dataset is the result of applying XOR function on the mask bit and the fingerprint bit.

Assume we want to fingerprint a very simple dataset shown in Table 3.2 with a fingerprint 11110000. We use the value 01010101 as a secret key $\mathcal{K}$. Furthermore, assume we want

Algorithm 3.2: AK Scheme: Detection Algorithm

Input: fingerprinted dataset $\mathcal{R}'$ with scheme $(P, A_0, ..., A_{v-1})$
Output: suspected buyer's ID id

//initiate fingerprint template and counts
fingerprint template $\mathcal{F} = (f_0, ..., f_{L-1}) = (?, ..., ?)$ // '?' represents unknown value
for $i = 0$ *to* $L - 1$ **do**
    $count[i][0] = count[i][1] = 0$
    // $count[i][0] = count[i][1]$ are votes for f_i to be 0 and 1 respectively
end
//scan all tuples and obtain counts for each fingerprint bit
foreach *tuple* $r \in R'$ **do**
    if $\mathcal{S}_1(\mathcal{K}, r.P)$ *mod* $\gamma == 0$ **then**
        attribute_index $i = \mathcal{S}_2(\mathcal{K}, r.P)$ mod v
        bit_index $j = \mathcal{S}_3(\mathcal{K}, r.P)$ mod v
        mark_bit $m = LSB(j, r.A_i)$
        mask_bit $x = 0$ if $\mathcal{S}_4(\mathcal{K}, r.P)$ is even; $x = 1$ otherwise
        fingerprint_bit $f = m \oplus x$
        fingerprint_index $l = \mathcal{S}_5(\mathcal{K}, r.P)$ mod L
        //update the votes
        $count[l][f] + +$
    end
end
//recover the fingerprint
for $l = 0$ *to* $L - 1$ **do**
    if $count[l][0] + count[l][1] == 0$ **then**
        return *none suspected*
    end
    $f_l = 0$ if $count[l][0]/(count[l][0] + count[l][1]) > \tau$
    $f_l = 1$ if $count[l][1]/(count[l][0] + count[l][1]) > \tau$
    return *none suspected* otherwise
end
$\mathcal{F} = (f_0, ..., f_{L-1})$
//determine a source of leakage
$id = detect(\mathcal{F}, \mathcal{K}, L, N)$
if $id \geq 0$ **then**
    return id
else
    return *none suspected*
end

Algorithm 3.3: AK Scheme: Subroutine *detect*

```
1 detect(template F, secret key K, fingerp. length L, number of buyers N):
2 for each buyer n do
3 |   F' = H(K|id)
4 |   if F == F' then
5 |   |   return id
6 |   end
7 |   return -1
8 end
```

Table 3.2: Sample dataset

Primary key	Attribute 0	Attribute 1
1	34	749
2	21	265

to mark on average every second tuple, i.e. $\gamma = 2$ and consider the last two bits of value for marking; $\xi = 2$. The algorithm will use the random number sequence generator to produce a unique sequence of numbers for every tuple (different seed for every tuple). Let

$$\mathcal{S}(01010101|0) = \{72, 39, 10, 34, 97\} \text{ and}$$
$$\mathcal{S}(01010101|1) = \{21, 37, 62, 25, 16\}.$$

Starting with the first tuple, the algorithm will choose it for marking because $\mathcal{S}_1 \bmod \gamma = 72 \bmod 2 = 0$. Then the attribute index is chosen as $\mathcal{S}_2 \bmod v = 39 \bmod 2 = 1$ and bit index as $\mathcal{S}_3 \bmod \xi = 10 \bmod 2 = 1$, therefore we are marking 2nd LSB (because bit indices start with 0) of a value 749 which is 0; $751_{10} = 1011101101_2$. Further, we decide the mark that is going to be applied to the bit. Firstly, the algorithm computes a mask bit as $\mathcal{S}_4 \bmod 2 = 34 \bmod 2 = 0$ and decides which fingerprint bit to use; $\mathcal{S}_5 \bmod L = 97 \bmod 8 = 1$, i.e. second bit of the fingerprint (indices start from 0) - 1. Secondly, the mark bit value is calculated as $mask\_bit$ xor $fingerprint\_bit = 0$ xor $1 = 1$, and finally the mark bit is embedded into previously chosen place in the data, i.e we are changing 2nd LSB of 749 from 0 to 1 obtaining that way a fingerprinted value 751. The algorithm continues with the second tuple. The condition from the line 3 of Algorithm 3.1 fails because $\mathcal{S}_1 \bmod \gamma = 21 \bmod 2 = 1$. Thus, the insertion algorithm does not mark this tuple and the process is over.

Detection Algorithm 3.2 shows the pseudo-code of fingerprint detection. The detection algorithm must reverse the steps from the fingerprint insertion phase to detect all the bits that construct a valid fingerprint. In the line 2 of the algorithm, the template for the fingerprint is initialised with L unknown values "?". Reversing the steps from the insertion phase is possible because the pseudo-random sequence generator $\mathcal{S}$ will produce the same random number sequences when seeded with the same value. Therefore, modelled by the

insertion algorithm, it is iteratively seeded with a concatenation of a secret key $\mathcal{K}$ known only to the owner and the primary key of every single tuple. In lines 9-12, based on random numbers from the sequence, the location of the marked bit is calculated. In the same manner, the mask bit x and fingerprint bit index i are calculated, in lines 13 and 15 respectively. Considering that in the insertion phase the value of a bit to be embedded, i.e. the mark bit m is calculated by applying the XOR function on the fingerprint bit and the mask bit, the fingerprint bit f is then, in reverse, calculated by applying the XOR function on the mark bit m and the mask bit x (line 14). Note that the fingerprinted data set might have been under attack that changes or erases the values from the originally released fingerprinted dataset, for example, subset attack or bit-flipping attack. This kind of attacks might disturb the fingerprint detection phase and the fingerprint bits might not be detected correctly. The record of detected values of a fingerprint bit f_l during the detection phase is kept in two count variables $count[l][0]$ and $count[l][1]$ depending if the detected bit is 0 or 1, respectively. When the counts for each fingerprint bit are obtained, the algorithm assigns 0 to fingerprint bit f_l if the counts satisfy the condition

$$count[l][0]/(count[l][0] + count[l][1]) > \tau \tag{3.2}$$

or 1 if counts satisfy the condition

$$count[l][1]/(count[l][0] + count[l][1]) > \tau \tag{3.3}$$

The parameter $\tau \in [0.5, 1)$ defines assurance of the detection process. Recovered fingerprint bits constitute the fingerprint template $\mathcal{F} = (f_0, ..., f_{L-1})$ which is compared to fingerprints of buyers to detect the source of leakage. This process is done by the subroutine *detect* described in Algorithm 3.3. The buyers' fingerprints are calculated on the fly. If the exact match of buyer n's fingerprint with the fingerprint template is detected, the buyer n is reported.

3.4.2 Assumptions and Properties

In this section, we list the assumptions and properties of AK Scheme. The first assumption is one that is present in all of the schemes presented in this thesis. We have to assume that the minor errors in numerical attributes in the dataset, necessarily caused by fingerprinting process, are not violating the integrity of the database and that those errors are tolerated by the database users. Besides, the AK Scheme is modelled such that it requires the presence of a primary key attribute. The primary key should stay unmodified or otherwise has to be recoverable for the sake of successful fingerprint detection. The same is true for the tuple order in the database. Since each tuple is assumed to have a unique primary key value based on which it is processed independently, the scheme is incrementally updatable. This means that fingerprint bits can be added to any additional tuples in the dataset at any given point in time in the future, without breaking the integrity of a fingerprinting scheme.

One more advantage of AK Scheme is blindness: It is not required to have the original database or any of the fingerprints involved in the fingerprint detection stored as the

owner's secret key is involved in every step of the scheme, both embedding and detecting a fingerprint.

3.5 Block oriented scheme

The block-oriented fingerprinting scheme for relational databases is very much inspired by the block oriented scheme in spatial domain proposed in [62]. This scheme divides the image to be fingerprinted into blocks of size $\beta \times \beta$ and permutes them in an order which is specific for every buyer. Both permutation and the information of the buyer are stored in a database known to the owner only. The scheme calculates the minimum and maximum intensities of the pixels in every block, and according to the corresponding bit of the fingerprint, increases (if the bit value is 1) or decreases (if the bit value is 0) intensities of the pixels in the block. In this way, every buyer gets a marked image which is different for everyone. Each marked image is different from the original, however, the changes are hardly perceptible to the human. This ability to produce imperceptibly different copies of the original does not apply in the same way in relational databases. This is one of the reasons why multimedia fingerprinting techniques cannot be directly applied to relational datasets.

In this section, the algorithms for the block-oriented scheme will be presented. Furthermore, we discuss the main properties and limitations of the scheme and present the analysis of quality effects of a process of embedding the fingerprint.

3.5.1 Algorithms

Insertion For fingerprint insertion, it is assumed that an input relational dataset contains primary key attribute P and v numerical attributes $A_0, ..., A_{v-1}$. The pseudo-code is shown in Algorithm 3.4. Every buyer has her identification number which we use for fingerprint embedding, and which is allowed to be publicly available, similar to the AK Scheme. The fingerprint of a fixed length L of a buyer n is generated using a cryptographic hash function $\mathcal{H}$ as a hash of a concatenation of owner's secret key $\mathcal{K}$ and n. Every buyer gets a unique value called *threshold* (denoted as r_0) which is used as a seed for a pseudo-random sequence generator in the insertion algorithm. The term *threshold* is used in the literature [2], however, it does not have any functionality of providing boundaries in the process as the naming would suggest. It is solely used as a seed for the pseudorandom number generator. To avoid storing the pairs of buyer's IDs and thresholds, in the adaptation of this scheme for this thesis we use a concatenation of owner's secret key and buyer's ID as a buyer's threshold value. The next step is to create the binary image using the bits of values available for embedding the fingerprint and dividing it into blocks of size $\beta \times \beta$. Table 3.3 shows the example of creating the binary image from the sample dataset and dividing the binary image into blocks. Table 3.3b shows the sample dataset from Table 3.3a with binary representation of its values. In this example we allow three LSBs of all the values to be the candidates for marking ($\xi = 3$). These bits are underlined in Table 3.3b. If the binary representation of an original value

Algorithm 3.4: Block Scheme: Insertion Algorithm

Input: dataset $\mathcal{R}$ with scheme $(P, A_0, ..., A_{v-1})$, buyer's ID n
Output: fingerprinted dataset $\mathcal{R}'$

fingerprint of buyer n: $\mathcal{F}(\mathcal{K}, n) = \mathcal{H}(\mathcal{K}|n)$
choose a threshold r_0 for the pseudo-random number generator
divide dataset attribute values bits into blocks B_i of size $\beta \times \beta$
$i = 0, j = 0$
foreach *block* B_i **do**
    $r_1 = \text{random}(r_0)$
    $x = H_1(r_1, n) \bmod \beta$
    $r_2 = \text{random}(r_1)$
    $y = H_1(r_2, n) \bmod \beta$
    $B_i(x, y) = B_i(x, y) \oplus f_j$
    $r_0 = r_2$
    $i++, j++$
    if j==L **then**
        j=0
    end
end
return $\mathcal{R}'$

does not reach ξ, the leading zeros are added. They are extracted to construct the binary image in Table 3.3c which is divided into blocks of size $\beta \times \beta$, here $\beta = 2$, shown in Table 3.3d. This blocked image serves as a background structure for embedding the fingerprint bits. Blocks are being marked in order such that the position (x, y) of a bit within the block i to be marked is generated by a pseudo-random number choice. The random number generator is always seeded by the previously generated number, with the threshold r_0 being a seed for the first generation. The next generated number r_1 The new value of the bit on the chosen position in the block is calculated as XOR of the original bit value and the fingerprint bit in the order. Fingerprint bits are embedded in sequential order and circularly, i.e. when the last fingerprint bit is embedded while there are still unmarked blocks left, the bits are being embedded all over from the start until all blocks are marked. This means that under assumption that the length of the fingerprint is of form $\mathcal{F} = [f_0, f_1, ..., f_3 1]$, blocks with the sequence number 0, 32, 64, ... will be marked with f_0, block with the sequence number 1, 33, 65, ... will be marked with f_1, etc.

Detection It is crucial to have the complete data in the suspicious database, therefore before blocking the bit image, it is necessary to properly order the tuples according to the primary key, as well as the attributes according to the original dataset and to fill out possibly missing tuples and attributes. To do so, the detection algorithm needs the access to the original dataset. The pseudo-code for the detection algorithm is shown in

Algorithm 3.5: Block Scheme: Detection Algorithm

Input: fingerprinted dataset $\mathcal{R}'$ with scheme $(P, A_0, ..., A_{v-1})$
Output: suspected buyer's ID n

sort $\mathcal{R}'$ according to the primary key P
divide bits of $\mathcal{R}'$ into blocks $\beta \times \beta$
foreach *buyer* n **do**
  retrieve the corresponding r_0
  $F_{n,j} = 0$ for all $j \in \{0, ..., L-1\}$
  $i = 0$
  foreach *block* B_i **do**
    $j = i \mathrm{mod} L$
    $r_1 = \mathrm{random}(r_0)$
    $x = H_1(r_1, n) \bmod \beta$
    $r_2 = \mathrm{random}(r_1)$
    $y = H_1(r_2, n) \bmod \beta$
    $F_{n,j}$ += $\mathcal{R}'(B_i(x,y)) \oplus \mathcal{R}(B_i(x,y))$ if $\mathcal{R}'(B_i(x,y))$ is in $\mathcal{R}$
    $r_0 = r_2$
    $i++$
  end
end
foreach *buyer* n **do**
  fingerprint of buyer n: $\mathcal{F}(\mathcal{K}, n) = \mathcal{H}(\mathcal{K}|n)$
  foreach $j \in \{0, ..., L\}$ **do**
    if $F_{n,j}/\omega \geq \tau$ **then**
      $f'_{n,j} = 1$
    else if $1 - F_{n,j}/\omega \geq \tau$ **then**
      $f'_{n,j} = 0$
    else
      $f'_{n,j} =?$
    end
  end
  if $\mathcal{F}_n == \mathcal{F}'_n$ **then**
    return *buyer* n *is the source of leakage*
  end
end
return *none suspected*

Table 3.3: Creating $\beta \times \beta$ blocks in binary image; $\beta = 2$, $\xi = 3$

(a) Original dataset

P	A_0	A_1	A_2	A_3
0	32	2	14	165
1	26	1	15	171
2	30	4	19	169
3	23	4	22	183

(b) Binary representation of the original values

P	A_0	A_1	A_2	A_3
0	100000	010	01110	10100101
1	011010	001	01111	10101011
2	011110	100	10011	10101001
3	010111	100	10110	10110111

(c) Binary image

000010110101
010001111011
110100011001
111100110111

(d) Binary image divided into blocks

00	00	10	11	01	01
01	00	01	11	10	11
11	01	00	01	10	01
11	11	00	11	01	11

Algorithm 3.5. Once the dataset is complete, the next step is to divide the bit image made of LSBs into blocks of the same size as in the insertion phase, $\beta \times \beta$. Then we retrieve the corresponding r_0 of each buyer that we use as an initial seed to random numbers generator. The detection algorithm repeats steps from the insertion phase to locate the bits that are marked. The algorithm finds a position (x, y) in each block B_i where the mark is embedded (lines 7-11 of Algorithm 3.5). Since the bits in the insertion phase are marked with the result of *xor* operation between the existing bit on the position (x, y) of the i-th block and j-th fingerprint bit, in the detection phase the operation is inverted to obtain the fingerprint bit value (line 12). Thus, the marked bit on the position (x, y) of B_i *xor* the original bit on the position (x, y) of B_i will yield the value of the fingerprint bit f_j. The variable $F_{n,j}$ counts how many times the fingerprint bit f_j was detected to be 1. At the end of the extraction process, the fingerprint value of the potential fingerprint $\mathcal{F}'$ is decided according to the values of count variables. If the value of a certain fingerprint bit is during the process counted to be 1 more than $\tau\omega$ times, then we set that fingerprint bit to be 1. The analogy holds for deciding that fingerprint bit is 0. The parameter $\tau \in [0.5, 1)$ is the assurance of the detection process. Choosing for example $\tau = 0.5$ for the detection algorithm means that if count variable of the fingerprint bit f_0 counted more than 10 occurrences of value 1 out of 20 actual embeddings of the fingerprint it f_0 in the dataset, the algorithm would pass the condition in line 21 and the extracted value of the fingerprint bit f_0 will be 1.

3.5.2 Assumptions and Properties

The most important parameter for the Block Scheme is β that defines the size of the block. Since every block is being marked in the insertion algorithm, β defines the robustness of the fingerprint and the distortion of data. In a dataset with v attributes, η tuples and ξ

LSBs available for marking, the number of blocks in the binary image of the dataset is:

$$\frac{v\xi}{\beta} \cdot \frac{\eta}{\beta} \tag{3.4}$$

The bits for marking are chosen pseudorandomly. In a dataset big enough and with a reasonable disparity of its values, we can assume a uniform distribution of values of the chosen bits. After a *xor* operation between a specific fingerprint bit and a bit to be marked, there is 50% chance the bit will change its original value. Thus, in the fingerprinted dataset the number of changed values will be

$$\frac{1}{2} \cdot \frac{v\xi}{\beta} \cdot \frac{\eta}{\beta} \tag{3.5}$$

One of the practical limitations occurs when creating blocks in the binary image of a dataset. Let us assume the dataset has 5 attributes, each allowing three LSB-s for mark embedding, i.e. $5 * 3 = 15$ bits available for marking in the row. If we choose $\beta = 4$, it will cause last three bits of the row never to be part of any block, hence, never marked. Table 3.4 depicts the disputable situation. The binary image of data is divided into three blocks of size 4×4 and the remainder of size 3×4 is left out. Generally, if possible, β should be a divisor of $v\xi$ (number of attributes times number of LSBs available for marking) to avoid such a situation. Hence, $\beta \leq v\xi$.

Table 3.4: Limitations in block creation

(a) Binary image

000010110101110
010001111011011
110100011001000
111100110111110

(b) Binary image divided into three blocks and the remainder

0000	1011	0101	110
0100	0111	1011	011
1101	0001	1001	000
1111	0011	0111	110

3.6 Two-level Fingerprinting technique

Guo et al. [3] propose a fingerprinting technique for protecting numerical relational data from illegal duplication and redistribution. The fingerprint embedding scheme contains two embedding processes. In the first embedding process, a unique fingerprint that identifies a specific buyer is embedded in relational data using a secret key known only to the owner of the database. The fingerprint can be detected using the same secret key to prove ownership at a numerical confidence level. The second embedding process is designed for verifying the extracted fingerprint. Thus, the scheme provides ownership identification and illegal distributor identification on two separate numerical confidence levels.

The scheme uses the primary key for the identification of each tuple and for embedding the fingerprint. The proposed technique is marking a single attribute that is predefined based on practical attribute properties, taking into account that the embedding algorithm introduces small distortions to the least significant bits of the values. The scheme-specific notation is shown in Table 3.5. For the rest of the notations we refer to Table 3.1.

Table 3.5: Notation in the Two-level Scheme

$1/\gamma_1$	Marked ratio in the first embedding process
$1/\gamma_2$	Marked ration in the second embedding process
α_1	Significance level of the ownership
α_2	Significance level of each fingerprint bit
α_3	Significance level of the fingerprint

3.6.1 Algorithms

Insertion The insertion (embedding) algorithm whose, pseudocode is shown in Algorithm 3.6, combines two embedding processes. The first process (lines 1-6) uses a cryptographic hash function to produce hash values of a concatenation of the secret key and primary key. The hash values are used to group tuples into L groups. Each group is associated with one fingerprint bit f_i. Since the hash results are uniformly distributed, each group is expected to have a similar number of tuples. The fingerprint bit f_i is part of a seed for a hash function that decides which tuples in the group i will be marked by f_i. We concatenate the fingerprint bit f_i, primary key $r.P$ and secret key $\mathcal{K}$, use it as a seed for the hash function and compute the modulo by γ_1. Due to the uniformity of a hash function, on average $\frac{1}{\gamma_1}$ is the fraction of tuples that will be selected for marking. A different seed, created by the same values permuted, is then used in a hash function that decides which LSB will be marked by f_i in the marking process. The described marking pattern is used to verify the ownership independently from the fingerprint.

The second embedding process (lines 7-17) for marking considers only the tuples that have not already been marked in the first embedding process to avoid overlapping. This process uses the fingerprint itself as a secret key. Similarly to the first process, the hash function results are used to select the tuples and LSBs to be marked. The selected bit is marked "0" if the hash result of secret key $\mathcal{K}$ concatenated with primary key $r.P$ is odd, otherwise "1". The granularity of the second embedding process is controlled by γ_2. The fraction of tuples marked in the second process is $(1 - \frac{1}{\gamma_1}) * \frac{1}{\gamma_2}$. Thus, the total fraction of tuples marked can be calculated as in Equation (3.6).

$$\frac{1}{\gamma} = \frac{1}{\gamma_1} + (1 - \frac{1}{\gamma_1}) * \frac{1}{\gamma_2} \tag{3.6}$$

Detection The fingerprint detection (fingerprint extraction) algorithm consists of three tasks:

Algorithm 3.6: Two-level Scheme: Insertion Algorithm

Input: dataset $\mathcal{R}$ with primary key P, buyer's fingerprint $\mathcal{F}$

```
1  foreach tuple r ∈ R do
2      i = H(r.P|K) mod L
3      group[i] ← r
4      if H(f_i|r.P|K) mod γ_1 == 0 then
5          j = H(r.P|K|f_i) mod ξ
6          LSB(j, r) = f_i
7      else if H(F|r.P|K) mod γ_2 == 0 then
8          j = H(r.P|K|F) mod ξ
9          if H(K|r.P) mod 2 == 0 then
10             LSB(r, j) = 0
11         end
12         else
13             LSB(r, j) = 0
14         end
15     else
16         do nothing to this tuple
17 end
```

Output: fingerprinted dataset $\mathcal{R}'$

1. Ownership verification
2. Fingerprint extraction
3. Fingerprint verification

The first task is to find the pattern to verify the ownership, the second is to extract the suspect's fingerprint and finally, the third task is verifying the extracted fingerprint. Algorithm 3.7 comprise the first two tasks - ownership verification and fingerprint extraction. Fingerprint verification is described by Algorithm 3.10. All parameters and the secret key should be the same as used in the embedding algorithm.

The first step of the extraction algorithm is ownership verification. We use the original secret key $\mathcal{K}$ to find the pattern embedded in the embedding process. Firstly, we use subroutine *detect* (Algorithm 3.8) to identify the candidate set of tuples for detecting marks. The candidate tuples are sorted into $subset_0$ and $subset_1$ depending on the conditions in lines 4 and 12 of the subroutine *detect*. If a tuple satisfies both conditions, it is included in both $subset_0$ and $subset_1$. $total\_count_0$ and $total\_count_1$ count the number of candidate tuples in $subset_0$ and $subset_1$, respectively. These steps ensure that all of the tuples selected for marking in the embedding process (Algorithm 3.6, line 4) are going to be selected as candidate tuples (although the inverse does not hold; there exist candidate tuples that were not chosen for marking in the first embedding process).

Algorithm 3.7: Two-level Scheme: Fingerprint Extraction Algorithm

Input: suspect relation $\mathcal{R}'$ with primary key P'

1 total_count$_0$, total_count$_1$, match_count$_0$, match_count$_1$ = *detect*($\mathcal{R}'$)
2 total_count = total_count$_0$ + total_count$_1$
3 match_count = match_count$_0$ + match_count$_1$
4 **if** *match_count > threshold(total_count,* α_1*)* **then**
5 **foreach** *tuple* $r \in \mathcal{R}$ **do**
6 $i = \mathcal{H}(r.P|\mathcal{K})$ mod L
7 group[i] $\leftarrow r$
8 **end**
9 **foreach** *group[i]* **do**
10 total_count$_0$, total_count$_1$, match_count$_0$, match_count$_1$ = *detect*(group[i])
11 **if** *match_count*$_0$*>threshold(total_count*$_0$*,*α_2*)* **then**
12 $f_i = 0$
13 **else if** *match_count*$_1$*>threshold(total_count*$_1$*,*α_2*)* **then**
14 $f_i = 1$
15 **else**
16 fail to extract f_i
17 **end**
18 **end**
19 **end**
20 **return** *fingerprint* $\mathcal{F}$

The total number of candidate tuples $total\_count = total\_count_0 + total\_count_1$ will be $\approx \eta/\gamma_1 + \eta/\gamma_1 = 2\eta/\gamma_1$.

Once the tuple is selected as a candidate, the algorithm checks whether the bit positions that were supposed to be marked in the embedding process in the candidate tuples are the correct values. In line 7 we calculate the same hash as in embedding process - $\mathcal{H}(r.P|\mathcal{K}|1)$ to obtain the bit position that should contain a mark, and in lines 8-10 we record if the mark is correct by the count variable $match\_count_1$. The same steps are done for candidate tuples from $subset_0$ in lines 15-18. The total number of matches from the candidate tuples is then $match\_count = match\_count_0 + match\_count_1$. Note that in unaffected fingerprinted data, we expect to see η/γ_1 matches that correspond to tuples marked in the fist embedding process. This algorithm further produces more "fake" matches, about $\eta/2\gamma_1$ of them, so in total the rough expectation is $match\_count = 3\eta/2\gamma_1$.

From this extraction phase, the ratio of matches and the total number of candidate tuples is $match\_count/total\_count = 75\%$. This forms a special pattern, and a probability to detect such a pattern in unmarked data is rather small. The authors propose a threshold

Algorithm 3.8: Two-level Scheme: Subroutine *detect*

```
1  detect(relation R):
2  total_count_0 = total_count_1 = match_count_0 = match_count_1 = 0
3  foreach tuple r ∈ R do
4      if H(1|r.P|K) mod γ_1 == 0 then
5          // subset_1
6          total_count_1++
7          j = H(r.P|K|1) mod ξ
8          if j^th bit is 1 then
9              match_count_1++
10         end
11     end
12     if H(0|r.P|K) mod γ_1 == 0 then
13         // subset_0
14         total_count_0++
15         j = H(r.P|K|0) mod ξ
16         if j^th bit is 0 then
17             match_count_0++
18         end
19     end
20 end
21 return total_count_0, total_count_1, match_count_0, match_count_1
```

Algorithm 3.9: Two-level Scheme: Subroutine *threshold*

```
1  threshold(n, α):
2  return minimum integer m that satisfies Σ_{k=m}^{n} c_n^k (1/2)^n < α
```

value which satisfies:

$$P\{MATCH\_COUNT > threshold | total\_count\} < \alpha \tag{3.7}$$

i.e. the probability to find matches more than the threshold is less than *alpha* in a non-marked relation, where $alpha \in (0, 1)$ is a small value called significance level. Thus, once such a pattern is detected, we can claim that the relation must have been modified by our insertion algorithm at the confidence level of $(1 - \alpha)$. The threshold (Algorithm 3.9) for a given *total_count* is calculated as Equation (3.8).

threshold = minimum integer m that satisfies

$$\sum_{k=m}^{n} c_n^k \left(\frac{1}{2}\right)^n < \alpha \tag{3.8}$$

where

$$c_n^k = \frac{n!}{k!(n-k)!}; n = total\_count \tag{3.9}$$

In line 4 of Algorithm 3.7 we compare the total number of matches with the threshold. If the number of matches is larger than the threshold, the ownership verification succeeded; otherwise, the ownership cannot be claimed.

When the ownership verification is done, the algorithm attempts to extract the fingerprint to track the buyer that leaked the data without the authorisation. The procedure starts with line 5 of Algorithm 3.7 and like the insertion algorithm forms the same L groups. The group represents the subset of tuples that are marked with the same fingerprint f_i. Therefore, for each group the candidate tuples and matches are calculated using again the subroutine *detect* (Algorithm 3.3) to extract the value of each fingerprint bit. For this phase, we use a significance level α_2. A fingerprint bit f_i is claimed to be 0 at confidence level $(1-\alpha_2)$ if the $match\_count_0$ is larger than the threshold calculated with the number of candidate tuples from the group that might contain mark 0 from the embedding phase - $total\_count_0$ and α_2. Following the analogy, fingerprint bit f_i is claimed to be 1 if the $match\_count_1$ is larger than the corresponding threshold. If neither $match\_count_0$ nor $match\_count_1$ exceed the corresponding threshold, the fingerprint bit value cannot be decided. The output of Algorithm 3.7 is an extracted candidate fingerprint which needs further verification.

Algorithm 3.10: Two-level Scheme: Fingerprint Verification Algorithm

```
   Input: suspect relation R', suspect fingerprint F'
 1 foreach tuple r ∈ R' do
 2     if H(F'|r.P|K) mod γ2==0 && H(f_i')|r.P|K mod γ1!=0 then
 3         total_count++
 4         j = H(r.P|K|F') mod ξ
 5         if H(K|r.P) is even && j^th bit is 0 then
 6             match_count++
 7         else if H(K|r.P) is odd && j^th bit is 1 then
 8             match_count++
 9     end
10 end
11 if match_count>threshold(total_count, α3) then
       Output: the fingerprint is verified
12 end
13 else
       Output: the fingerprint is not verified
14 end
```

Using the Algorithm 3.10, we identify the exact fingerprint from the candidate set of suspect fingerprints produced by Algorithm 3.7. In this phase, we detect the embedding

pattern from the second phase of fingerprint embedding algorithm at a confidence level of $(1-\alpha_3)$. Following the steps of the second phase of the fingerprint insertion algorithm, we consider only the tuples that have not been marked in the first embedding process. The pattern is detected using the owner's secret key $\mathcal{K}$ and the candidate fingerprint $\mathcal{F}$. The Algorithm 3.10 in line 3 counts the candidate tuples that are supposed to be marked, and depending on the hash value and value of the corresponding fingerprint bit, counts the matches in lines 6 or 8, depending on whether the bit value is 0 or 1. Note that in this phase if the data is unaffected, the parameters and the secret key are the same as in the insertion algorithm, and also the correct fingerprint is extracted in the previous phase, then the number of candidate tuples *total_count* and number of matches *match_count* is expected to be equal. The number of matches is once again compared to the threshold calculated from *total_count* and significance level α_3. If the number of matches satisfies the condition in line 11, we may claim that the fingerprint $\mathcal{F}$ is verified at confidence level $(1-\alpha_3)$.

3.6.2 Properties and discussion

We mentioned in the previous section that the extraction algorithm relies on finding a certain number of matches that would confirm the mark pattern is embedded in the data. The confidence levels of the fingerprint extraction process (i.e. the converse process to the first embedding processes) are defined by significance level parameters α_1 and α_2 that are used to calculate the *threshold* value. To achieve a high confidence level of ownership, e.g. 99% ($\alpha_1 = 0.01$), the condition in line 4 of Algorithm 3.7 has to be satisfied.

Figure 3.1 shows, for a fixed $\alpha_1 = 0.01$, the thresholds for different values of *total_count* as a fraction of *total_count* (continuous blue line). The dashed blue line is the expected fraction of *match_count* out of *total_count* (75%). We can see that for the *total_count* larger than ≈ 30 the previously mentioned condition is satisfied and the extraction algorithm can verify the ownership with a confidence level of 99%. The statement is confirmed with the experimental results shown by the orange lines in the figure. The experiments are run on Forest Cover Type data with fingerprint length $L = 96$. The *match_count* value in the experiments is $75\%(total\_count) \pm 5\%$, according to the expectation. Furthermore, the lower limit for *total_count* to have 99% confidence level for ownership is shown to be around 30. Therefore, Equation (3.10) needs to be satisfied for the correct ownership verification.

$$2\eta/\gamma_1 > 30(\alpha_1 = 0.01) \tag{3.10}$$

For the real-life datasets with hundreds or thousands of tuples this is rather easy to achieve (e.g. for Forest Cover Type data, setting $\gamma_1 = 200$ results in $total\_count \approx 5,800$).

The second phase of detection algorithm - the fingerprint extraction - identifies each bit individually from the associated group of tuples. The process of comparing the matching bits to the threshold value is in this phase controlled by α_2. If the number of matches

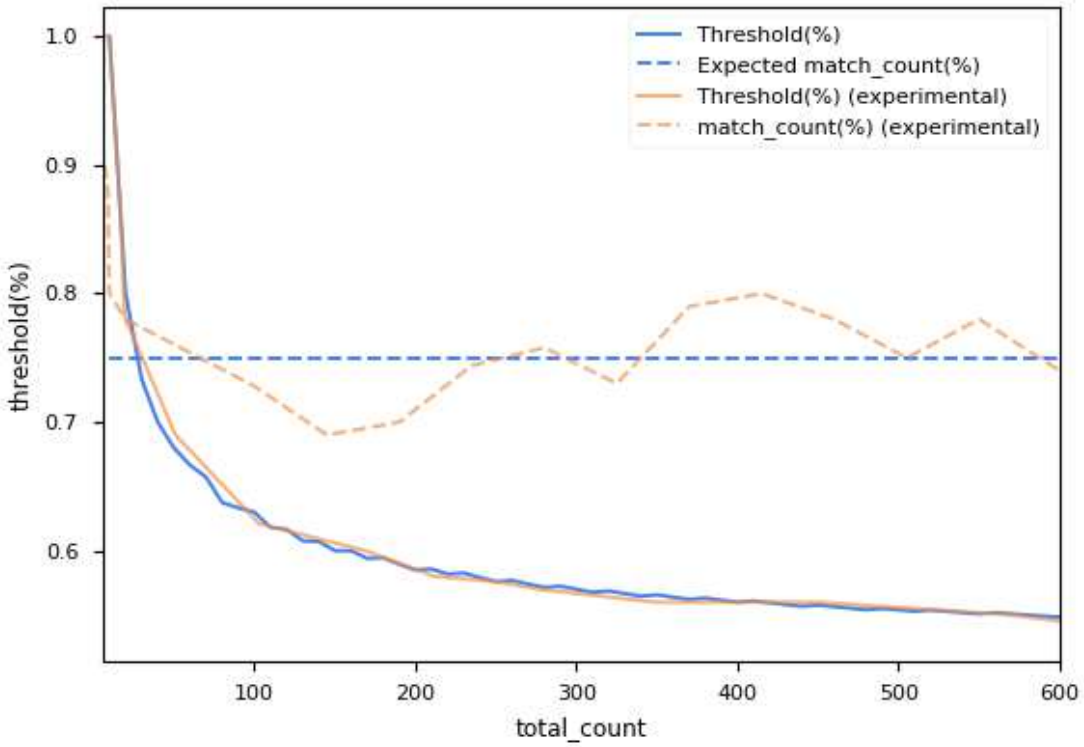


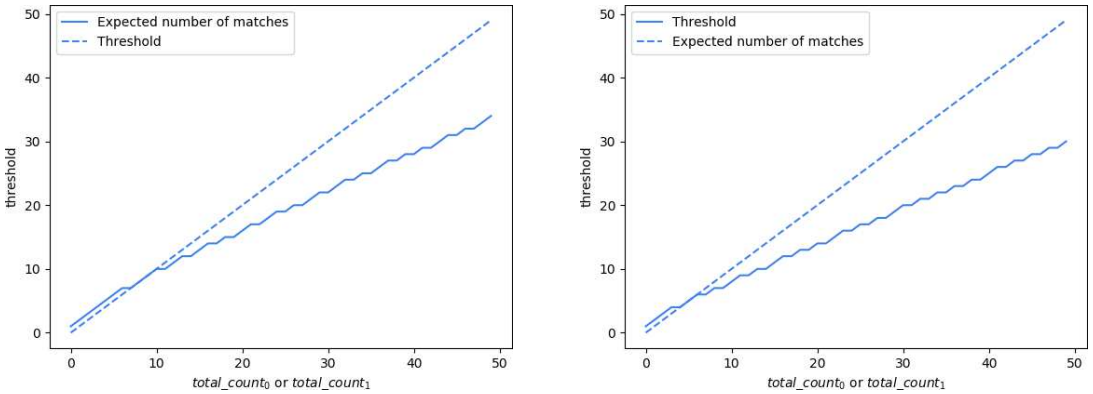
Figure 3.1: Portion of match_count and *threshold* for different total_count to achieve the ownership confidence level 99% ($\alpha_1 = 0.01$)

of a single bit value is larger than the threshold, the bit is decided to be that value (conditions in lines 11 and 13 of Algorithm 3.7).

Figure 3.2 shows threshold values in a subset, depending on the total number of tuples in a subset. A dashed line represents the expected number of matches in a marked group of unaffected fingerprinted data. We show the relation between thresholds and the number of matches for different levels of confidence - 99% and 90%. Generally, the more tuples are selected into a subset, the more robust the fingerprint is. For reaching the confidence level of 99%, $total\_count_i$ must be at least 8 for a trusted pattern to be found, while for 90% confidence that lower limit is 4. Therefore, Equation (3.11) must be satisfied to obtain the successful marking pattern with confidence 99% and Equation (3.12) with confidence 90%.

$$\eta/(\gamma_1 * L) > 7(\alpha_2 = 0.01) \tag{3.11}$$

$$\eta/(\gamma_1 * L) > 3(\alpha_2 = 0.1) \tag{3.12}$$



(a) $\alpha_2 = 0.01$ (b) $\alpha_2 = 0.1$

Figure 3.2: Threshold in subsets of unaffected marked data for different $total\_count_i$ to achieve confidence level of each bit of 99% (a) and 90% (b)

In the real implementation $\eta/(\gamma_1 * L)$ should be rather larger than suggested in Equations (3.11) and (3.12) because of the deviation introduced by random distribution of tuples into L groups.

The proposed technique has a limitation that it embeds the fingerprint in only one, initially chosen attribute. However, this approach is vulnerable to a vertical attack (attribute deletion attack) where the attacker can delete the fingerprint by removing the entire attribute. To avoid the possibility of such attack in multi-attribute datasets and for further experiments with this scheme, we slightly redesign the embedding and extraction process such that all attributes appear as candidates for marking. The simple modification is the reinterpretation of the term $LSB(r, j)$ in Algorithm 3.6. In this notation $LSB(r, j)$ represents the $j - th$ bit from the set of least significant bits available for marking of *all* attribute values from the tuple r (instead of only one predefined attribute). Pseudo-randomness of this step ensures uniformly distributed marks over all attributes.

3.7 Fingerprinting categorical data

We present two approaches to developing a technique that can be applied to both numerical and categorical data. Both schemes use the processes of the AK Scheme for marking numerical values. The first technique, a rather simplistic one, is based on a random choice of values to be marked and a random choice of a mark. In the second approach, we focus on keeping the semantic relations between the categorical data rather than purely randomly choosing the marks. The technique, however, relies on the availability of the original data, therefore, unlike the first scheme, it is not blind.

Another proposed technique for fingerprinting categorical data is [66], where the k-anonymity pattern is used as a fingerprint. The number of k-anonymity patterns is limited and highly depends on the data and user-defined generalisation hierarchies, therefore also the number of distinct fingerprints is limited. Furthermore, k-anonymity significantly changes the quality of the data. Since a subset of values is replaced by the respective value of the higher semantic category (e.g. Vienna->Austria), the informativeness of the data is lost. Every buyer receives a distinct fingerprinted data copy, thus the quality of the data among the distributed copies differs significantly. The technique lacks in implementation, therefore we do not include it in our analysis.

3.7.1 Random mark choice

Methodology

Insertion The approach for fingerprint embedding into the categorical data has the following steps:

1. Label encoding all categorical values to obtain numerical data
2. Fingerprint insertion algorithm of AK Scheme (Algorithm 3.1)
3. Applying modular arithmetic to fingerprinted numerical values of categorical data
4. Decoding the values of categorical attributes

The main idea of the approach is to convert categorical values to numerical values and treat them as any of the described methods for fingerprinting numerical data would. Assume that the attribute A has c different categorical values $C_0, ..., C_{c-1}$. The label encoding model assigns each categorical value C_i to the respective numerical value i. After the fingerprint is embedded, the numerical values are being decoded back to the categorical values. A problem arises when a fingerprinted numerical value is not assigned to any of the categorical values in the encoding model because the range of values available in fingerprinting 2^{ξ}, due to the binary representation, does not always correspond to the range of categorical values for an attribute. The occurrence of such illegal values is possible when the fingerprint insertion algorithm marks a bit in the numerical representation of categorical value in such a way that the change of that bit transforms the numerical value to something that can't be decoded. For instance, assume that an attribute A_1 has 6 different categorical values ($c = 6$), assigned by the encoding model to the numerical values $\{0, 1, 2, 3, 4, 5\}$. Furthermore, assume $\xi = 2$, and the insertion algorithm deciding to change the second least significant bit of value "4" somewhere within the dataset. The bit representation of the value $4_{10} = 100_2$ will after marking be changed to 110_2 which corresponds to 6 in the decimal system, the value that is not assigned to any categorical value of attribute A_1. In general, the set of values of A_1 that can be obtained after embedding the fingerprint is $\{0, 1, 2, 3, 4, 5, 6, 7\}$, where 6 and 7 are illegal values.

Much bigger odds for a marked value to be out of bounds is in cases when the number of LSB-s available for fingerprinting ξ is bigger than the length of bit representation the numerical values in label encoding model. For example, assume $c = 4$, i.e. an attribute A_2 has 4 different values that are assigned to numerical values $\{0, 1, 2, 3\}$, and $\xi = 3$. In case when insertion algorithm chooses, for example, value $2_{10} = 10_2$ and its third least significant bit for fingerprinting, the resulting marked value is $110_2 = 6_{10}$, which does not have a corresponding categorical value in the encoding model. In general, after the fingerprint-embedding process, the domain of possible values of the attribute A_2 is $\{0, .., 7\}$, where $\{4, 5, 6, 7\}$ are not assigned to any original categorical value in our label encoding model.

We solve the problem of illegal values in the fingerprinted dataset by applying modular arithmetic to the set of values obtained after fingerprinting. The final numerical value x'_i of the attribute is given by $x'_i = x_i \ mod \ c$, where x_i is the value after step 2 of the fingerprint-embedding process. The modulo step in insertion algorithm for categorical data applied on A_1 would change the marked value $6_{10} = 110_2$ to $6_{10} \mod 6 = 0_{10} = 000_2$.

Finally, after removing the illegal values from the fingerprinted dataset, the numerical values are decoded to the categorical values.

Detection Fingerprint detection process contains the following steps:

1. Label encoding categorical values using the same model as in fingerprint insertion phase
2. Fingerprint detection algorithm of AK Scheme (Algorithm 3.2)

In the fingerprint detection process, it is again necessary to preprocess data by label encoding categorical values. The encoding model needs to match the one from the insertion phase, otherwise, the detection process is disrupted. After encoding categorical values, the AK detection algorithm 3.2 can be directly used to detect the malicious buyer.

Properties and Discussion

The properties of a detection algorithm for categorical data are essentially the same to the properties of AK detection algorithm, except that it is important to take into account the effects of modular arithmetic applied to the fingerprinted values where it is the case. Consider the previous example with attribute A_1 where the fingerprint insertion algorithm changes the number representation $2_{10} = 10_2$ of some categorical value to value $110_2 = 6_{10}$ by marking the third LSB and modulo operator application obtains the final marked value $6_{10} \mod 6 = 0_{10} = 000_2$. From this attribute value, the fingerprint bit cannot be correctly extracted anymore, therefore the detection algorithm is affected.

The detection algorithm uses the technique called *majority voting* for deciding the values of fingerprint bits, as discussed in Section 3.4. This is the key to the property of the

scheme that small errors in data do not prevent detecting the source of leakage. The effects of these small errors, i.e. values that are changed after inserting the fingerprint, are analysed in the context of attacks in Chapter 4. Applying modulo to the values has the same effect on the detection process as purposely changing the values in fingerprinted data as part of a malicious attack (for example, bit flipping attack).

The success of the detection process directly depends on the number of values that are a result of performing the modulo step in the insertion phase. This number is affected by the number of least significant bits we allow to change (parameter ξ) - larger ξ leads to more frequent usage of modulo operation because the fingerprinted values more frequently get out of valid bounds. Since the detection process relies on multiple embeddings of a single fingerprint bit, having more marks in the dataset might assure that the fingerprint bits are detected correctly. This is controlled by parameter γ and the length of a fingerprint L.

Table 3.6 presents the success of the detection algorithm for different values of parameters γ and ξ. A fingerprint is in these experiments embedded in categorical values only. Length of a fingerprint L is set to 40. The dataset used for experiments from Table 3.6 is the German Credit data with 1000 entries and 21 attributes, out of which 14 are categorical. The dataset is described in Section 4.1.2. The results are based on 1000 runs of the algorithm for every parameter combination.

Table 3.6: Success of a detection algorithm of the fingerprinting scheme for categorical data using the German Credit dataset

	$\xi = 1$	$\xi = 2$	$\xi = 4$	$\xi = 6$
$\gamma = 2$	99.8%	99.3%	56.4%	23.2%
$\gamma = 3$	94.7%	90.4%	17.3%	2.9%
$\gamma = 6$	30.0%	18.2%	0.3%	0%
$\gamma = 9$	2.9%	1.0%	0%	0%
$\gamma = 12$	0.1%	0%	0%	0%

The detection algorithm performance significantly drops when either ξ or γ are increased. The expectation that the performance of the detection algorithm drops if more LSB-s are available for embedding the fingerprint (larger ξ) is confirmed by these experiments. Table 4.2 shows how many categorical attributes have a certain amount of different values in German Credit dataset. For instance, 3 out of 14 categorical attributes have 3 different values and therefore require only 2 bits to be encoded. An additional 10 attributes have up to 5 different values, therefore can be encoded by 3 bits.

In the case where our insertion algorithm marks the 1st or the 2nd LSB of the value, the value obtained after fingerprinting will most likely be a modified numerical value that can be decoded back to a valid categorical value. However, allowing e.g. the 4th significant bit to be marked in the insertion process opens more possibilities of obtaining values outside of the bounds, and modulo needs to be applied. This results in the big decrease

in the detection that can be seen in Table 3.6 between $\xi = 2$ and $\xi = 4$. Applying modulo changes the marked value in the way that the detection algorithm can't extract the fingerprint bit correctly, leading to the impossibility of detecting a valid fingerprint. Generally, increasing ξ leads to worse performance of the detection algorithm. A good choice for value of parameter ξ depends very much on the dataset. One needs to inspect the dataset to see approximately how many different values the categorical attributes have, and set ξ accordingly. For example, if most of the categorical attributes have approximately 4 values, then ξ should be at most 2 to keep the performance of the detection algorithm high. Generally, ξ should be set to at most the number of bits needed to encode all the values of the attribute to their binary representations. The choice of parameter ξ according to this will help in decreasing the loss in performance of the detection algorithm, but it generally cannot be completely avoided. One of the reasons is that the choice of ξ is limited to choosing the same value for the entire dataset, i.e. the same number of LSBs is available for fingerprinting in any of the attributes, no matter the size of their domains. The optimal choice for ξ from a detection performance point of view is then constrained by attributes with a small number of different values, while from the robustness point of view it is desired to have larger ξ, as discussed in detail in Chapter 4. On the other hand, if the numbers of different values in all categorical attributes are not all power of two and ξ not set according to the size of the smallest domain, there will always be even a minor possibility that the *modulo* will be applied and will affect the detection algorithm.

Besides the parameter ξ, the parameter γ affects the performance of the detection algorithm, as shown in Table 3.6 ($\gamma = 2$ means that on average every second tuple is chosen for fingerprinting). Experiments show a significant decrease in performance when increasing γ. Increasing γ means marking fewer tuples. Consequently, each fingerprint bit will be embedded in the data fewer times. This makes it harder for the detection algorithm to extract the correct value of a specific fingerprint bit when errors caused by the modulo operation are introduced. The design of our detection algorithm requires the perfect match of the fingerprint extracted by detection algorithm to some valid buyer's fingerprint, therefore only one falsely extracted fingerprint bit is enough for detection algorithm to fail. This is the reason for the success of 0% for experiments with both high γ and high ξ.

The issue that arises when fingerprinting dataset as small as German Credit data with only 1000 rows is that with γ large enough, some fingerprint bits don't get embedded at all in the insertion process. Let us have an example where $\gamma = 12$ and $L = 40$. According to the parameters, $1000/\gamma \approx 83$ rows will be fingerprinted, and each fingerprint bit will be embedded into the dataset $83/40 \approx 2$ times. This number is approximate, and most importantly, averaged over all fingerprint bits. Since the choice of fingerprint bit to be embedded is completely independent and random in every step of the insertion algorithm, it is plausible to expect some bits being embedded 0 times. This means that the detection algorithm will fail to extract the valid fingerprint from the unchanged fingerprinted dataset if the insertion process failed to embed all fingerprint bits at least once. This

observation is important to be noted when analyzing results from the Table 3.6 because the success of the detection algorithm in small datasets is affected not only by errors caused by modulo operation but also by the failure of insertion algorithm to embed all of the fingerprint bits for larger γ. To analyze these effects as separate cases, we define the following measures:

- *DFR* (Detection Fail Rate) - the number of fingerprint bits wrongly extracted because of the errors caused by modulo operation in the fingerprint detection process
- *IFR* (Insertion Fail Rate) - the number of fingerprint bits that were not embedded at all in the fingerprint insertion process, and are therefore impossible to be extracted (unknown bit value)
- *DIFF* - bit difference between the extracted fingerprint and the correct one; $DIFF = DFR + IFR$

Table 3.7 shows the rates obtained by the experiments shown in the Table 3.6, and presents a comparison of how much effect on the success of detecting the valid fingerprint is due to the modulo operation, and how much due to non-embedded bits. Rates in the table are the average values of rates from all 1000 runs. Insertion Fail Rate (*IFR*) depends only on the number of dataset values being fingerprinted, i.e. parameter γ. For γ as small as two, a single fingerprint bit is embedded $1000/(\gamma * L) \approx 13$ times on average, and experimental results show that the insertion algorithm does not fail in embedding all of the fingerprint bits in any of the 1000 trials. The detection algorithm is, for $\gamma = 2$, affected only by errors induced by modulo, and the fingerprint extracted by the algorithm and the correct one differ in only 0.002 bits on average. For $\gamma = 12$ the *IFR* raises up to $\approx 5/40$, i.e. on average 5 out of 40 bits of the extracted fingerprint are incorrect. From the experiments, we can see that in most of the cases DFR is larger than IFR, i.e. errors in fingerprint extraction are mostly caused by applying modulo operation, except for small values of ξ.

Table 3.7: Bit difference and the detection fail rates for the German Credit data

	$\xi = 1$		$\xi = 2$		$\xi = 4$		$\xi = 6$		
	$DIFF$	DFR	$DIFF$	DFR	$DIFF$	DFR	$DIFF$	DFR	IFR
$\gamma = 2$	0.002	0.002	0.007	0.007	0.562	0.562	1.376	1.376	0
$\gamma = 3$	0.055	0.045	0.100	0.090	1.654	1.644	3.203	3.193	0.010
$\gamma = 6$	1.170	0.558	1.642	1.030	5.603	4.991	7.842	7.230	0.612
$\gamma = 9$	3.615	1.105	4.256	1.746	9.206	6.696	11.498	8.988	2.510
$\gamma = 12$	6.462	1.424	7.190	2.152	11.935	6.897	14.095	9.057	5.038

Table 3.8 shows the experimental results of the success of the detection algorithm when modulo is not applied, but illegal numerical values are left in the fingerprinted dataset.

Experiments are as well run on the German Credit dataset, with 1000 runs for each parameter value. This table highlights the case of having a small dataset. The results show how limited the choice of parameter γ is in these cases.

Table 3.8: Success of a detection algorithm using the German Credit dataset without applying modulo operation

$\gamma = 2$	$\gamma = 3$	$\gamma = 6$	$\gamma = 9$	$\gamma = 12$	$\gamma = 15$
100%	99.0%	54.5%	7.6%	0.6%	0%

We repeat the experiments on another, larger dataset in order to examine the effects of modulo operation on the detection process, without other side effects such as the insertion algorithm failure. For the experiments, we use Adult dataset. The dataset originally has 32,561 rows that contain missing values. We will not examine dealing with missing values, so for the experiments, we use the subset of 30,162 rows that do not contain any missing values. Table 3.9 shows the experimental results of the success of the detection algorithm to recognise the correct malicious buyer. the fingerprint length L is set to 80. The results in Table 3.9 are the percentage of successfully detected fingerprints out of 1000 runs for every parameter combination.

Table 3.9: Success of the detection algorithm using Adult dataset

	$\xi = 1$	$\xi = 2$	$\xi = 4$	$\xi = 6$
$\gamma = 3$	100%	100%	100%	100%
$\gamma = 6$	100%	100%	100%	100%
$\gamma = 12$	100%	100%	100%	99.5%
$\gamma = 25$	100%	100%	95.8%	64.0%
$\gamma = 50$	95.6%	72.6%	25.2%	1.4%
$\gamma = 100$	14.2%	3.0%	0%	0%

Comparing the results from Table 3.6 and Table 3.9 it can be observed that the detection algorithm performance improved a lot by having a larger dataset. For low γ the performance is perfect or close to perfect. Again, increasing ξ leads to worse performance of the detection algorithm since the modulo operation is required more and fingerprint bits are wrongly detected. In Table 3.10 we can see that the extracted fingerprint is very "close" to the valid one. For instance, in the case of $\gamma = 50$ and $\xi = 2$, the performance of the algorithm drops to 72.6%, but the bit difference between the extracted and real fingerprint is on average 0.31 (Table 3.10, $DIFF$ rate), meaning that for most of the remaining 27.4% runs of the algorithm, the fingerprints' bit difference is only in one single bit. In Table 3.10 we omitted the experiments where the detection success is 100% since all the rates for those cases are 0. In other cases, we can see that insertion fail rate IFR is very small or zero, so most of the fingerprint detection failure is caused by modulo operation.

The problem of having illegal categorical values after the fingerprinting process is thwarted

Table 3.10: The detection fail rates for Adult data

	$\xi = 1$		$\xi = 2$		$\xi = 4$		$\xi = 6$		
	$DIFF$	DFR	$DIFF$	DFR	$DIFF$	DFR	$DIFF$	DFR	IFR
$\gamma = 25$	0	0	0	0	0.043	0.043	0.431	0.431	0
$\gamma = 50$	0.047	0.003	0.310	0.266	1.302	1.258	3.987	3.943	0.044
$\gamma = 100$	1.994	0.158	4.086	2.250	7.030	5.194	12.228	10.392	1.836

by the solution that introduces another problem - the weaker performance of the detection algorithm. Bad performance of the detection algorithm necessarily means that the presented fingerprinting scheme for categorical data is more susceptible to the attacks. Small datasets are more vulnerable because of the possibility that the insertion algorithm fails to embed the fingerprint properly, not only in the context of categorical data but in general. By careful dataset inspection and parameter tuning, both problems of small dataset and errors caused by modulo can be avoided. Lower L, γ and ξ in general lead to better performance.

We saw that the number of LSB-s ξ defines the quality of the scheme and that by choosing smaller ξ we can minimise the error of the detection algorithm. In the discussion above, we use the fixed ξ for all the attributes. However, we could modify this property such that each attribute A_i is associated with own ξ_i based on the number of distinct values of that attribute. This way we increase the robustness by setting particular ξ_i values as high as possible, taking into account the upper limit that is defined by the number of distinct values in the attribute. For instance, changing only one LSB in the attribute with only three values, but allowing to change 5 LSB-s in attributes with > 40 values.

This approach is simplistic but effective. We will see in Chapter 4 the robustness analysis of the scheme and in Chapter 5 the utility analysis. Another aspect that could be taken into account when designing a robust fingerprint technique is the semantic correlation between the categorical attributes. For instance, the fingerprint might be much more perceptible when there is a frequent number of occurrences of impossible or very unlikely combinations of attribute values due to marking. This problem is addressed in the following section.

3.7.2 Choosing a mark based on correlations in the dataset

This approach addresses the problem of semantic relations between categorical attributes that can be disturbed by fingerprinting. Considering attributes independently of each other and embedding a random mark into a categorical value might lead to non-consistent records. A mark may introduce an uncommon or impossible combination of values in the data. As an example, let us consider a dataset containing attributes *gender*, *numberOfPregnancies*, etc. The attributes *gender* and *numberOfPregnancies* intuitively contain an impossible combination of values: (*gender*:male, *numberOfPregnancies*:1).

Another example is where the combination of values might be very uncommon. Take, for example, a medical dataset containing information about the patients suffering from Alzheimer's disease. The combination (*alzheimersStage*:middle, *employed*:yes) is very uncommon, but might be introduced by a random fingerprint mark. With a dataset domain knowledge, these examples would be rather suspicious and thus perceptible. We aim to take into account the correlation between the values of different attributes and avoid uncommon combinations. Algorithm 3.11 shows the pseudocode for the insertion algorithm of the scheme.

Methodology

Algorithm 3.11: Fingerprinting technique for categorical data: Insertion Algorithm

Input: dataset $\mathcal{R}$ with scheme $(P, A_0, ..., A_{v-1})$, buyer n's ID id
Output: fingerprinted dataset $\mathcal{R}'$

fingerprint of buyer n: $\mathcal{F}(\mathcal{K}, id) = \mathcal{H}(\mathcal{K}|id)$
foreach *tuple* $r \in \mathcal{R}$ **do**
  if $(\mathcal{S}_1(\mathcal{K}|r.P)\ mod\ \gamma == 0)$ **then**
    attribute_index $i = \mathcal{S}_2(\mathcal{K}|r.P)$ mod v
    if A_i *is categorical* **then**
      mask_bit $x = 0$ if $\mathcal{S}_3(\mathcal{K}|r.P)$ is even; $x = 1$ otherwise
      fingerprint_index $l = \mathcal{S}_4(\mathcal{K}|r.P)$ mod L
      fingerprint_bit $f = f_l$
      mark_bit $m = x \oplus f$
      if $m == 1$ **then**
        neighbourhood = *select_neighbours*()
        target_values, freq = *get_frequencies*(neighbourhood)
        $r.A_i$ = *random*(target_values, *weight* = freq)
      end
    else if A_i *is numerical* **then**
      bit_index $j = \mathcal{S}_3(\mathcal{K}|r.P)$ mod ξ
      mask_bit $x = 0$ if $\mathcal{S}_4(\mathcal{K}|r.P)$ is even; $x = 1$ otherwise
      fingerprint_index $l = \mathcal{S}_5(\mathcal{K}|r.P)$ mod L
      fingerprint_bit $f = f_l$
      mark_bit $m = x \oplus f$
      $LSB(j, r.A_i) = m$
  end
end
return $\mathcal{R}'$

Insertion The insertion algorithm resembles the AK Scheme's insertion algorithm (Algorithm 3.1). The creation of a fingerprint and the pseudorandom choice of tuples and attributes to be marked is the same. In this scheme, the distinction is made based

on whether a numerical or a categorical attribute is chosen for fingerprinting (line 5). In a case where the attribute is categorical, the next random value generated by a pseudorandom number generator $\mathcal{S}$ decides the value of a mask bit x. Furthermore, the next random value from the generator decides which fingerprint bit f_l is going to be embedded. The mark bit m is a result of an XOR operation between the mask and the fingerprint bit. In case the mark is 1, the attribute value will be marked. The lines 11 to 13 of Algorithm 3.11 are specific for this scheme and contain the main part of this scheme. Instead of marking the value to something random from the domain of the attribute, the idea is to choose a value taking into account the values of the other attributes in the tuple. This way, the algorithm avoids the combinations of values that are unlikely to appear in the dataset. We search for a neighbourhood of the observed tuple regarding all attributes but one that is being fingerprinted. We find the neighbours using the nearest neighbours algorithm with the Hamming distance. We let the user select whether the neighbourhood will be defined as a certain number of neighbours, k, or as the set of elements within the given distance d. The parameters k and d are predefined by the user as well. After the neighbours are obtained, we observe the values in the attribute A_i and sort them by their frequencies. The new value, i.e. the fingerprinted value is then a random value from the set of neighbours' values, where the random choice is weighted by values' frequencies. This technique is known in genetic algorithms as a fitness proportionate selection, or roulette wheel selection, a genetic operator used for selecting potentially useful solutions for recombination. Fingerprinting process of numerical values follows the steps of the AK Scheme.

Detection The pseudocode for the detection algorithm is shown in Algorithm 3.12.

Similarly to the AK's detection algorithm (Algorithm 3.2), the algorithm starts by initializing the fingerprint template and the votes for fingerprint bit values. We then find the tuples and attributes that should have been fingerprinting according to the same pseudorandom sequence as in the insertion algorithm. For categorical attributes, the fingerprint extraction is described from line 7 to 12. We retrieve the mask bit x. Next, to find the value of the mask bit m, it is necessary to compare the suspected fingerprinted dataset to the original. If the corresponding value is different from the original, then the value m was 1 in the insertion algorithm, otherwise 0. Furthermore, we find the fingerprint bit index l as the next random value in the pseudorandom sequence generator. The lines 13 to 17 contain the extraction from the numerical values which is the same as in the extraction algorithm of the Ak Scheme. The lines 18 to 38 are common for both categorical and numerical data and follow the steps from the detection algorithm of the AK Scheme.

Discussion

Neighbourhood search is implemented in two different ways and is left to the user to decide which one to use for the specific case. We allow searching for a fixed number of neighbours, k, or searching for neighbours within a predefined distance d. The choice between the

Algorithm 3.12: Fingerprinting technique for categorical data: Detection Algorithm

Input: fingerprinted dataset $\mathcal{R}'$ with scheme $(P, A_0, ..., A_{v-1})$, original dataset $\mathcal{R}$ with scheme $(P, A_0, ..., A_{v-1}$

Output: suspected buyer's ID id

fingerprint template $\mathcal{F} = (f_0, ..., f_{L-1}) = (?, ..., ?)$
$count[i][0] = count[i][1] = 0$ **for** $i = 0$ to $L - 1$
foreach *tuple* $r \in R'$ **do**
  if $\mathcal{S}_1(\mathcal{K}, r.P)$ *mod* $\gamma == 0$ **then**
    attribute_index $i = \mathcal{S}_2(\mathcal{K}, r.P)$ mod v
    if A_i *is categorical* **then**
      mask_bit $x = 0$ if $\mathcal{S}_3(\mathcal{K}, r.P)$ is even; $x = 1$ otherwise
      if $r.A_i$ *is different from the original* **then**
        mark_bit $m = 1$
      else
        mark_bit $m = 0$
      fingerprint_index $l = \mathcal{S}_4(\mathcal{K}, r.P)$ mod L
    else if A_i *is numerical* **then**
      bit_index $j = \mathcal{S}_3(\mathcal{K}, r.P)$ mod v
      mark_bit $m = LSB(j, r.A_i)$
      mask_bit $x = 0$ if $\mathcal{S}_4(\mathcal{K}, r.P)$ is even; $x = 1$ otherwise
      fingerprint_index $l = \mathcal{S}_5(\mathcal{K}, r.P)$ mod L
    //update the votes
    fingerprint_bit $f = m \oplus x$
    $count[l][f] + +$
  end
end
//recover the fingerprint
for $l = 0$ *to* $L - 1$ **do**
  if $count[l][0] + count[l][1] == 0$ **then**
    return *none suspected*
  end
  $f_l = 0$ if $count[l][0]/(count[l][0] + count[l][1]) > \tau$
  $f_l = 1$ if $count[l][1]/(count[l][0] + count[l][1]) > \tau$
  return *none suspected* otherwise
end
$\mathcal{F} = (f_0, ..., f_{L-1})$
$id = detect(\mathcal{F}, \mathcal{K}, L, N)$
if $id \geq 0$ **then**
  return id
else
  return *none suspected*
end

approach, as well as setting the parameters k and d requires some expert knowledge about the dataset. In case of an approach based on selecting a fixed number of neighbours, it is important to handle the neighbours with the same distances deterministically. We solve this in the following way: first, we choose the k neighbours, then find the maximum distance within the neighbours and select all elements outside of the neighbourhood with the same distance. Therefore, it might be the case where the neighbourhood is extended to more than k elements.

The set of attributes included in the neighbourhood selection do not have to be the whole set of attributes of the dataset. This can be set by the user who has insights into the data and has the knowledge about the highly related attributes.

The final choice of the mark could be ambiguous if we would always choose the most frequent ones, in cases when we have multiple values with the same frequencies. For that reason we choose to select the new value pseudorandomly, weighted by the frequencies. This way the whole set of values from the neighbourhood have a chance to be selected, while the most frequent one will be selected with the highest probability.

3.8 Summary

In this chapter we have presented three common fingerprinting techniques for fingerprinting relational datasets, the AK Scheme in Section 3.4, Block Scheme in Section 3.5 and Two-level Scheme in Section 3.6. These techniques have in common the usage of cryptographically secure structures and algorithms, i.e. cryptographic pseudo-random sequence generator and cryptographic hash function. They are used for creating the buyers' fingerprints because of security reasons since they must remain secret to everyone except the owner of the dataset. These techniques are limited to application on numerical values in the data. Furthermore, two fingerprinting techniques for non-numerical data are introduced in Section 3.7. Both techniques extend the AK Scheme such that the same algorithmic steps are used for fingerprinting the numerical part of the data. The first scheme follows the pseudo-random pattern of choosing marks for categorical values. In the second scheme, the solution goes towards preserving the correlations between the categorical values and marks the values in a way that no uncommon combinations of values occur in the final fingerprinted copy of the dataset.

These techniques are the basis of the analysis in the following chapters. The techniques are susceptible to attempts of a malicious buyer to destroy the fingerprint from the dataset. In the next chapter, we analyse how robust these techniques are under certain types of attacks.

CHAPTER 4

Robustness of fingerprinting techniques against attacks

4.1 Experimental setup

4.1.1 Robustness measures

Fingerprinting schemes should be robust against different attempts to prevent the correct detection of the fingerprint. Modifying, deleting and adding the values to the fingerprinted data, that can be both benign updates and malicious attacks, can modify or erase the fingerprint. A robust fingerprinting scheme should make it hard for the attacker to erase the fingerprint, to modify it in the way that an innocent buyer is implicated as a traitor, or to modify unmarked data such that a valid fingerprint is detected.

In further sections we analyse robustness fingerprinting schemes against different attacks using robustness measures proposed in [1]:

- **Misdiagnosis false hit** (fh^D): The probability of detecting a valid fingerprint from data that has not been fingerprinted.
- **Misattribution false hit** (fh^A): The probability of detection an incorrect but valid fingerprint from fingerprinted data.
- **False negative** (fn): The probability of detecting no valid fingerprint from fingerprinted data.
- **False miss** (fm): The probability of failing to detect an embedded fingerprint correctly. False miss rate is the sum of the false negative and misattribution false hit rates, i.e. $fm = fh^A + fn$.

4.1.2 Datasets

We use three different datasets for the experiments on fingerprint robustness and quality effects on fingerprinted datasets in the following sections obtained from the UCI Machine Learning repository [79]:

- Forest Covertype dataset[1]
- German Credit Data[2]
- Adult dataset (train data)[3]

Forest Cover Type The dataset contains measurements related to the forest cover originally obtained from US Geological Survey (USGS) and US Forest Service (USFS) data. This dataset is chosen due to its desired properties of containing multiple integer-valued attributes, because of its size and because its intended purpose is the classification problem with target *Cover_Type*. This dataset is often used for experiments in watermarking and fingerprinting literature [35, 1], therefore using this dataset gives the possibility of comparing our results with previous work. The dataset has 581,012 rows, each with 54 attributes and no primary key. For fingerprint insertion, an extra attribute - *id* is added to serve as the primary key, since the chosen fingerprinting techniques require the presence of the primary key for fingerprint embedding. 44 out of a total of 54 attributes of the dataset contain binary values. We use the remaining 10 integer-valued attributes for embedding fingerprints. Binary attributes are a result of one-hot encoding, therefore changing one value from 0 to 1 or vice versa would require changing another attribute's value to retain the structure. The binary attributes of Forest dataset are excluded because of this high correlation that defines different properties compared to the numerical values in the context of fingerprinting. In Table 4.1, the information about the attributes is given.

German Credit Data The dataset describes persons by attributes containing some personal information and classifies them as good or bad in terms of risk for the credit defaulting. The dataset has 1000 rows, 20 attributes and one target attribute. 13 out of 20 attributes in this dataset, as well as the target attribute in this dataset, are categorical and we use this dataset to analyse the performance of the fingerprinting scheme for categorical data described in Section 3.7 and for experiments on the impact of fingerprinting to a learning task in Section 5.2. The dataset is also chosen for its size. Because it is considerably smaller than Forest Cover Type, there is a possibility that the size of a dataset can affect the performance of fingerprinting algorithms. Detailed information about the attributes in the dataset is given in Table 4.2.

[1] https://archive.ics.uci.edu/ml/datasets/covertype
[2] https://archive.ics.uci.edu/ml/datasets/statlog+(german+credit+data)
[3] https://archive.ics.uci.edu/ml/datasets/adult

Table 4.1: Attributes of the Forest Cover Type dataset

Name	Description	Type
Elevation	Elevation in meters	Int
Aspect	Aspect in degrees azimuth	Int
Slope	Slope in degrees	Int
Horizontal_Dist_To_Hydrology	Horizontal distance to nearest surface water features	Int
Vertical_Dist_To_Hydrology	Vertical distance to nearest surface water features	Int
Horizontal_Dist_To_Roadways	Horizontal distance to nearest roadway	Int
Hillshade_9am	Hillshade index at 9am, summer solstice	Int
Hillshade_Noon	Hillshade index at noon, summer solstice	Int
Hillshade_3pm	Hillshade index at 3pm, summer solstice	Int
Horizontal_Dist_To_Fire_Points	Horizontal distance to nearest wildfire ignition points	Int
Wilderness_Area (4 columns)	Wilderness area designation	Binary
Soil_Type (40 columns)	Soil Type designation	Binary
Cover_Type	Forest Cover Type designation	Int

Adult dataset The dataset contains information about people such as their age, workclass, education, etc. and a target attribute whether a person makes at least $50K per year or less than $50K. This dataset contains 15 attributes in 30,162 samples of training data (after removing samples containing missing values), where the attributes are both numerical and categorical (five continuous numerical and ten categorical). The same as German Credit dataset, we use this dataset to analyse the performance of fingerprinting scheme for categorical data in Section 3.7 and for experiments on the impact of fingerprinting to a learning task in Section 5.2. Table 4.3 describes the attributes of the Adult dataset.

4.1.3 Attacks

In the following sections we discuss and analyse the robustness of the fingerprinting schemes, described in Chapter 3, against the following attacks:

- Subset attack - the attacker releases only a subset of the dataset rows as an attempt to erase the fingerprint
- Superset attack - the attacker adds additional rows to the original dataset and mixes them with the original rows
- Bit-flipping attack - the attacker flips arbitrary bits in the dataset in an attempt to erase the fingerprint
- Additive attack - the attacker inserts additional fingerprint to the data, claiming the ownership
- Collusion attack - multiple buyers compare their dataset copies and create a new copy in an attempt to erase the fingerprint of every buyer in the collision

Table 4.2: Attributes of the German Credit dataset

Name	Description	Type	#values
Attr.1	Status of existing checking account	Categ.	4
Attr.2	Duration in month	Int	
Attr.3	Credit history	Categ.	5
Attr.4	Purpose	Categ.	11
Attr.5	Credit amount	Int	-
Attr.6	Savings account/bonds	Categ.	5
Attr.7	Present employment since	Categ.	5
Attr.8	Installment rate of disposable income (%)	Int	-
Attr.9	Personal status and sex	Categ.	5
Attr.10	Other debtors/guarantors	Categ.	3
Attr.11	Present residence since	Int	-
Attr.12	Property	Categ.	4
Attr.13	Age in years	Int	-
Attr.14	Other installment plans	Categ.	3
Attr.15	Housing	Categ.	3
Attr.16	Number of existing credits at this bank	Int	-
Attr.17	Job	Categ.	4
Attr.18	Number of liable people	Int	-
Attr.19	Telephone	Bin	2
Attr.20	Foreign worker	Bin	2
Target	Good or bad credit risks	Bin	2

4.2 Misdiagnosis false hit

In this section, we will analyse the misdiagnosis false hit rate of different fingerprinting schemes. This robustness measure differs from the others in a way that it does not measure the success of a malicious attack or benign updates on the dataset. In contrast to the ability of the detection algorithm to detect the correct fingerprint from the pirated (and fingerprinted) data, the fingerprinting scheme may also, purely by chance, extract a valid but incorrect fingerprint from unmarked data. This phenomenon is measured by misdiagnosis false hit rate.

4.2.1 AK Scheme

Assume that the detection algorithm from the unmarked data extracts a potential fingerprint $f = (f_0, ..., f_{L-1})$, i.e. some bit string of length L. Furthermore, assuming that a single fingerprint bit f_i is extracted from the dataset multiple times, it is decided to be a single value (0 or 1) if that value is extracted more than $\tau\omega_i$, where ω_i is the number of times f_i is extracted. Due to the use of pseudo-random mask bits in this scheme, each time f_i is extracted, it will be extracted as 0 or 1 with probability 0.5,

Table 4.3: Attributes of the Adult dataset

Name	Type	#values
age	Continuous	-
workclass	Categorical	8
fnlwgt	Continuous	-
education	Categorical	16
education-num	Continuous	16
marital-status	Categorical	7
occupation	Categorical	14
relationship	Categorical	6
race	Categorical	5
sex	Categorical	2
capital-gain	Continuous	-
capital-loss	Continuous	-
hours-per-week	Continuous	-
native-country	Categorical	41
income	Categorical	2

which is modelled as an independent Bernoulli trial. Once when the detection algorithm is done processing the dataset, the probability of the value of one fingerprint bit f_i of the extracted potential fingerprint f being 0 is $B(\lfloor \tau\omega_i \rfloor; \omega_i, 0.5)$, and the same probability stands for f_i being 1. Therefore, the algorithm detects the potential fingerprint with the probability $\prod_{i=0}^{L-1} 2B(\lfloor \tau\omega_i \rfloor; \omega_i, 0.5)$. The probability that the extracted fingerprint is matching one of the N valid ones equals to choosing N bit strings out of 2^L possible ones: $N/2^L$. Now the overall misdiagnosis false hit rate is

$$fh^D = \frac{N}{2^L} \prod_{i=0}^{L-1} 2B(\lfloor \tau\omega_i \rfloor; \omega_i, 0.5) \tag{4.1}$$

and after 2^L cancels out

$$fh^D = N \prod_{i=0}^{L-1} B(\lfloor \tau\omega_i \rfloor; \omega_i, 0.5) \tag{4.2}$$

The misdiagnosis false hit rate is exponentially dependant on length of the fingerprint L. The rate can be reduced by increasing L. Table 4.4 shows the misdiagnosis false hit rate under different values of L and $\omega_i \approx \{100, 50\} : \forall i \in \{0, ..., L-1\}$, where $N = 100$ and $\tau = 0.5$ are fixed values.

We can see that for $L \gg log(N)$ we can almost completely avoid the misdiagnosis false hit ($fh^D \simeq 0$) and by this rule, we will be choosing the values for L in further analysis and experiments in this thesis.

Table 4.4: Misdiagnosis false hit rate for the AK Scheme

L	8	16	32	64	128
$fh^D(\omega_i = 100)$	0.7208	0.0052	2.70×10^{-7}	7.30×10^{-16}	5.31×10^{-33}
$fh^D(\omega_i = 50)$	0.9151	0.0084	7.01×10^{-7}	4.92×10^{-15}	2.42×10^{-31}

4.2.2 Block Scheme

The probabilistic model for the misdiagnosis false hit is the same as in the case of AK Scheme, so for analysis on this topic, we refer to Section 4.2.1.

4.2.3 Two-level Fingerprinting Scheme

Two-level Fingerprinting Scheme detects the detected fingerprint in multiple phases. The extraction starts with ownership verification followed by fingerprint extraction and fingerprint verification. The detection algorithm goes to the extraction phase only if the ownership is verified, suggesting that first, we have to take into account the probability of passing the first phase of detection.

Let us assume that the input to the detection algorithm is an unmarked dataset of size η. We set $\alpha_1 = \alpha_2 = \alpha_3 = 0.01$ and choose arbitrary γ_1 and γ_2. The subroutine *detect* 3.3 counts the amount of tuples that should have been marked with 1 ($total\_count_1$) and with 0 ($total\_count_0$) in the embedding process. Due to uniformly distributed output of the hash function $\mathcal{H}$, the approximate value of both $total\_count_1$ and $total\_count_0$ will be:

$$total\_count_1 \approx total\_count_0 \approx \eta/\gamma_1 \tag{4.3}$$

Furthermore, the subroutine counts number of matches of chosen bits with the suggested values ($match\_count_1$ and $match\_count_0$). Again, assuming the uniformity of the hash function:

$$match\_count_1 \approx total\_count_1/2 \tag{4.4}$$

$$match\_count_0 \approx total\_count_0/2 \tag{4.5}$$

To pass the ownership verification test, the total number of matches has to satisfy the condition:

$$match\_count > threshold(total\_count, \alpha_1) \tag{4.6}$$

where $match\_count = match\_count_1 + match\_count_0$ and $total\_count = total\_count_1 + total\_count_0$. Therefore, for the ownership verification with confidence 99% the following has to hold:

$$\frac{threshold(total\_count, 0.01)}{total\_count} < 0.5 \tag{4.7}$$

The value of this portion is shown in Figure 3.1 with the solid blue line. Even for very big $total\_count$, the threshold portion very slowly approaches 0.5. Thus, the ownership is very unlikely to be falsely verified in the unmarked dataset.

The extraction algorithm does not continue if ownership is not verified, therefore the scheme is extremely robust against misdiagnosis false hit.

4.2.4 Fingerprinting scheme for categorical data

The probabilistic model for the misdiagnosis false hit for the fingerprinting scheme for the categorical data is the same as in the case of AK Scheme. For the analysis, we refer to Section 4.2.1.

4.3 Subset attack

In the attempt to erase the fingerprint from the dataset, the attacker may release only a subset of tuples of a fingerprinted dataset. This is called a subset attack. In our attack model, we assume the attacker selects each tuple independently with probability p to include it in the pirated dataset. We also assume no other updates on the dataset are applied and no other attacks performed.

4.3.1 AK Scheme

A subset attack succeeds when all embedded bits for at least one fingerprint bit are deleted. Assuming that each fingerprint bit f_i is embedded ω_i times, then the probability that all embedded bits for f_i are deleted is $(1-p)^{\omega_i}$. The probability that no valid fingerprint will be detected from the dataset is then

$$fm = 1 - \prod_{i=0}^{L-1} (1 - (1-p)^{\omega_i}). \tag{4.8}$$

Table 4.5 shows the probability of a successful attack for different parameter γ values. $p' = 1 - p$ denotes the probability that a single tuple is deleted, i.e. the approximate percentage of deleted tuples since the choice of deletion is made independently by the attacker. We set $\eta = 581,012$, $v = 10$ (according to the properties of Forest Cover Type dataset that we use in empirical evaluation), $\xi = 4$ and $L = 96$. We can see from the table that the subset attack only gets to the reasonable level of probability for success with more than at least 90% deleted tuples, depending on γ. We have to take into account that as few as 1% of the tuples in this example is around 5810 tuples, which for the attacker might still be the acceptable amount of tuples to release without authorisation and perform the successful subset attack if γ is set high enough ($\gamma \geq 25$). In those cases where p' is large, γ should be set to the smaller value, since the probability for a successful subset attack decreases when γ decreases for the same p'. Therefore, we adapt γ to prevent the subset attack.

Experiments To present empirically the success of the subset attack, we performed the attack on the Forest dataset with $\eta = 581,012$ and $v = 10$ using different parameter

Table 4.5: Probability of a successful subset attack on the AK Scheme

	$p' = 70\%$	$p' = 80\%$	$p' = 90\%$	$p' = 95\%$	$p' = 99\%$
$\gamma = 6$	0	0	0	0	0.0038
$\gamma = 12$	0	0	0	5.6881×10^{-10}	0.4555
$\gamma = 25$	0	0	8.1088×10^{-10}	0.0004	0.99985
$\gamma = 50$	0	0	0.0003	0.1761	1
$\gamma = 100$	4.877×10^{-8}	0.0001	0.1586	0.9892	1

settings. The parameters are chosen the same way as shown in table 4.5 to be able to compare theoretical results with the empirical. The experimental results are shown in table Table 4.6. Every experiment is run 500 times and parameters are set as presented in the table. We set $L = 96$ and $\xi = 4$ where the later does not affect the success of the subset attack.

We can see from the table Table 4.6 that the results roughly match our analysis. The best rate of success has the attacks where most of the tuples are deleted (>95%) and the percentage of fingerprinted tuples is low (γ is high). Therefore, we can argue that the AK Scheme is robust against subset attacks.

Table 4.6: Experimental results of a subset attack success rate on the AK Scheme, using the Forest dataset

	$p' = 70\%$	$p' = 80\%$	$p' = 90\%$	$p' = 95\%$	$p' = 99\%$
$\gamma = 6$	0	0	0	0	0.004
$\gamma = 12$	0	0	0	0	0.5
$\gamma = 25$	0	0	0	0	1.0
$\gamma = 50$	0	0	0.002	0.194	1.0
$\gamma = 100$	0	0	0.20	0.9975	1.0

Note that our original dataset did not have a primary key. Instead, we added the attribute Id to serve as the primary key. For simplicity purposes, the Id values in the experiments are represented by the sequence number of the tuple. Furthermore, although during subset attack the attacker removes some tuples, we assume that the primary key of every preserved tuple will not change. In case the primary key is removed or manipulated, the recreation of one is crucial for the defence against the subset attack. Availability of the original dataset simplifies the process of recreating the primary key. A simple matching algorithm can be applied to the suspect dataset that compares values of bit positions which are not being selected for marking in the fingerprinting process to the same set of bit positions in the original data. This approach might be flawed by having multiple primary key value candidates for a single tuple. In that case, the additional decision step based on the similarity of the bits used for marking can be applied. Alternatively, the virtual primary key construction technique proposed in [36] can be used to generate the primary keys and does not require the presence of the original dataset. Primary

keys are generated from the unmarked parts or data and the owner's secret key using a cryptographic hash function. Without knowing the secret key, recreating the primary key values is unfeasible.

4.3.2 Block Scheme

For the Block Scheme detection algorithm, it is crucial to have the same number of tuples and attributes and their correct order in the suspicious database. Otherwise, we cannot detect a valid fingerprint. When the attacker removes the chosen tuples, the defence has to replace the deleted one with the corresponding ones from the original dataset. In our analysis we formulate the deletion of each tuple as an independent trial with two possible outcomes whose probabilities remain the same through the trials (*Bernoulli trial*). Furthermore, with $B(k; n.p)$ we denote the probability of having at least k successes in n trials with probability p of success. The detection algorithm will be able to detect the correct fingerprint if every fingerprint bit f_i occurs at least $\lfloor \tau\omega_i \rfloor$ times in the suspicious dataset. It means that at least $\lfloor (1-\tau)\omega_i \rfloor$ embedded bits for some fingerprint bit have to be deleted for the attack to succeed. If the attacker examines each tuple independently and selects it for inclusion in the pirated database (i.e. deletes it with probability $p' = 1-p$), the probability that the fingerprint bit f_i cannot be detected is $B(\lfloor (1-\tau)\omega_i \rfloor; \omega_i, p')$. Note that each fingerprint bit in the Block Scheme is embedded either ω or $\omega - 1$ times, so we approximate this value to ω for the convenience. Then the probability that the detection algorithm will fail to extract the fingerprint is

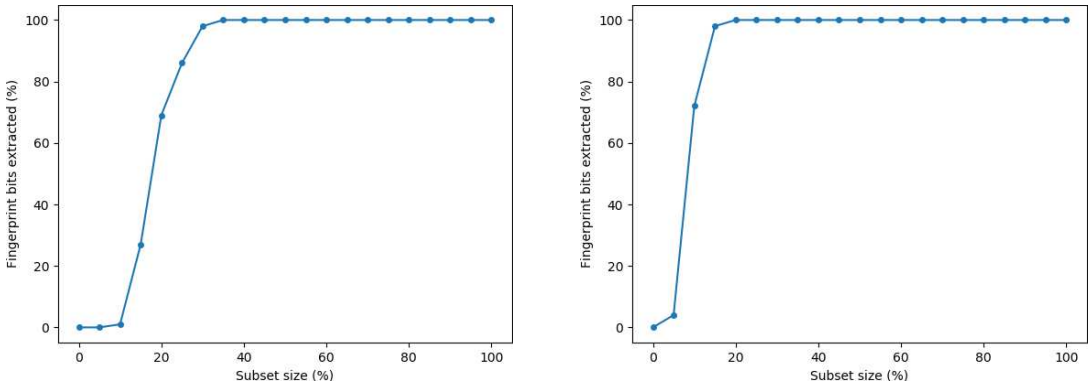
$$fm = 1 - (1 - B(\lfloor (1-\tau)\omega_i \rfloor; \omega_i, p'))^L \tag{4.9}$$

Table 4.7 shows the probabilities of successful subset attack under different values of parameter β and different probabilities p'. For calculations we use dataset of size $\eta = 581,012$ and $v = 10$ attributes, for purpose of comparing these results with experimental results. The other parameters are set as follows: $\xi = 3$, $L = 96$ and $\tau = 0.5$.

Table 4.7: Probability of a successful subset attack on the Block Scheme

	$p' = 30\%$	$p' = 40\%$	$p' = 45\%$	$p' = 50\%$
$\beta = 5$	0	0	0	1.0
$\beta = 10$	0	0	0.001	1.0
$\beta = 15$	0	6.8233×10^{-7}	0.2320	1.0
$\beta = 20$	0	9.7949×10^{-4}	0.8301	1.0
$\beta = 30$	2.0832×10^{-7}	0.2151	0.9998	1.0

Experiments We present the empirical results of the success of a subset attack on the Block Scheme. The experiments are run on Forest dataset of size $\eta = 581,012$ and $v = 10$ attributes, with parameters $\xi = 3$, $L = 96$ and $\tau = 0.5$. We measure success of the attack for $p' = \{0.30, 0.40, 0.45, 0.50\}$ and $\beta = \{5, 10, 15, 20, 30\}$. We run each experiment



(a) $\gamma_1 = 100$ (b) $\gamma_1 = 50$

Figure 4.1: Fingerprint extraction in a subset attack

500 times and summarise the number of successes/fails of the attack. In table 4.8 the empirical results are presented.

Table 4.8: Probability of a successful subset attack on the Block Scheme

	$p' = 30\%$	$p' = 40\%$	$p' = 45\%$	$p' = 50\%$
$\beta = 5$	0	0	0	1.0
$\beta = 10$	0	0	0.008	1.0
$\beta = 15$	0	0	0.356	1.0
$\beta = 20$	0	0.002	0.914	1.0
$\beta = 30$	0	0.196	1.0	1.0

For β values 5, 10 and 15, up to 40% of the arbitrary tuples can be deleted and it will not affect the detection of the fingerprint. Larger values of β tolerate around 30% of the deleted tuples. Compared to the theoretical results in Table 4.7, the experimental results are very similar.

4.3.3 Two-level Fingerprinting Scheme

The Two-level Fingerprinting scheme provides two levels of marking patterns; the first one that can verify the owner and the second one verifying the recipient of the dataset. When the attacker removes a subset of tuples and releases the remainder, the fingerprint is affected in two ways:

1. It might be impossible to trace back the owner but the owner can claim the ownership

2. It might be impossible both to trace back the owner and to claim the ownership

Experiments The attack may cause that the extraction process does not manage to extract all the fingerprint bits correctly. We can see from Figure 4.1(a) that even if only around 35% of the tuples are released by the malicious buyer, all fingerprint bits are still correctly extracted. The robustness of this scheme can be improved by changing the value of γ_1. Figure 4.1(b) shows the results for a smaller γ_1, i.e. for the increased number of marks in the data that can be used to verify the ownership. For this parameter setting, the scheme is more robust against the subset attack, and even from only 20% of the data, all the correct fingerprint bits can be extracted. For this experiments, we used the fingerprint of length $L = 96$ and the bit significance level of each bit α_2 is 0.01 (the confidence level is 99%).

Table 4.9: Success of the subset attack on Two-level Fingerprinting Scheme

	$p' = 60\%$	$p' = 70\%$	$p' = 80\%$	$p' = 90\%$	$p' = 95\%$	$p' = 99\%$
$\gamma_1 = \gamma_2 = 10$	0	0	0	0	0	1.0
$\gamma_1 = \gamma_2 = 25$	0	0	0	0.04	1.0	1.0
$\gamma_1 = \gamma_2 = 50$	0	0	0.04	0.98	1.0	1.0
$\gamma_1 = \gamma_2 = 100$	0	0.74	1.0	1.0	1.0	1.0
$\gamma_1 = \gamma_2 = 200$	1.0	1.0	1.0	1.0	1.0	1.0

Let us consider the setting where $\gamma_1 = 50$ and $\gamma_2 = 50$. For $p < 20\%$ ($p' > 80\%$; the attacker removes more than 80% of the tuples) some bit positions will be unknown. For $p = 15\%$ on average 2 out of 96 bit positions cannot be detected, creating 2^2 fingerprint candidates. The fingerprint verification process may still select the correct fingerprint out of 2^2 possible ones and the extraction algorithm finds the malicious buyer successfully. In the same setting, when 90% of the tuples are removed, the algorithm correctly extracts 72% correct fingerprint bits, or 69 out of 96. This is not enough for the fingerprint verification algorithm to verify any fingerprint and the detection algorithm fails. See the Table 4.9 where the success of the subset attack is recorded. The experiments are run using Forest Cover Type data. The results in the table confirm that although not all fingerprint bits are extracted, the attack is not 100% successful.

The first level of the embedding process provides the ownership verification even for cases when the correct fingerprint cannot be extracted. So, although the owner cannot trace the source of unauthorised leakage of data, she can claim the ownership. The mark created in this process is very robust. See in the Figure 4.2 that even from the very small percentage of the data (<5%), the detection algorithm will verify the owner with confidence level 99% ($\alpha_1 = 0.01$). The experiments are run even for the very big values of γ_1. For the settings where our experiments show high probabilities of extracting the right fingerprint from the small chunks of data, i.e $10 < \gamma_1, \gamma_2 < 100$, up to 99% of the tuples can be deleted and ownership can still be claimed.

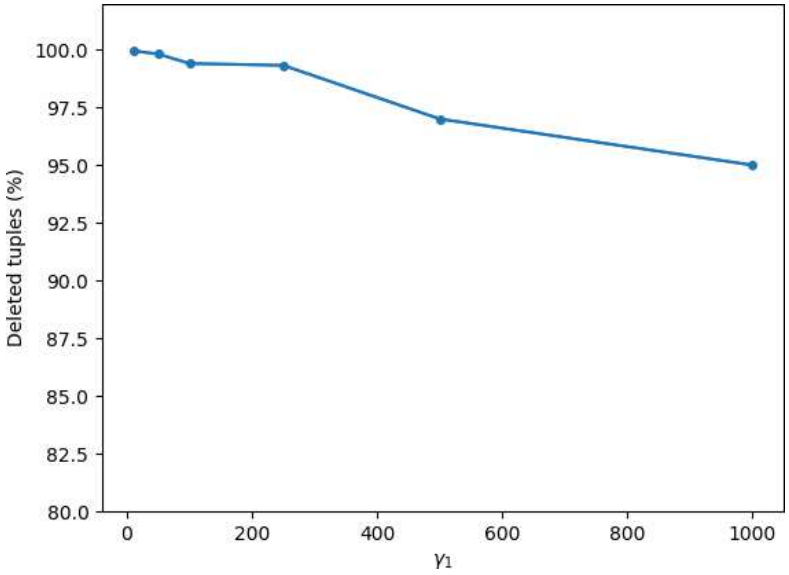


Figure 4.2: Ownership verification in a subset attack

4.3.4 Fingerprinting scheme for categorical data

The fingerprinting scheme for categorical data complements the AK Scheme and for subset attack, we can refer to Equation (4.8) as a starting point for analysis. The important difference between AK Scheme and scheme for categorical data is the introduction of modulo operation as one of the steps. We mentioned before in Section 3.7 that we trade the strength of detection algorithm for fingerprinting categorical data successfully. This means that additional modulo operation step in the fingerprint insertion phase causes errors in the detection phase that cannot be avoided. Having errors in unaffected fingerprinting scheme increases the vulnerability of the scheme to attacks. Therefore, the Equation (4.8) lacks influence of modulo operation in order to be credible. Table 4.10 shows the theoretical success of the subset attack on the dataset with $\eta = 30,162$ rows (convenient for comparison to experimental results on Adult dataset) if the effects of modulo are not taken into account, using Equation (4.8). ω_i is approximated to $\eta/(\gamma * L)$ and $L = 80$. Value 0 represents the perfect resistance to the subset attack, and 1 is the perfect success of the subset attack, i.e. the scheme completely failing to defend against it.

Experiments The experiments are made on the Adult dataset as it is the dataset containing categorical attributes. We measure the success of subset attack over 500 runs and parameters set as follows: $L = 80$, $\xi = 1$, $\tau = 0.5$, $\gamma = \{3, 6, 12, 25, 50, 100\}$ and $p' = \{0.30, 0.60, 0.80, 0.90, 0.95, 0.99\}$, where p' represents the percentage of tuples that are deleted.

Table 4.10 and Table 4.11 share the same parameter settings for calculating the success

Table 4.10: Theoretical success rate of a subset attack on the fingerprinting scheme for categorical data, using the Adult data

	$p' = 30\%$	$p' = 60\%$	$p' = 80\%$	$p' = 90\%$	$p' = 95\%$	$p' = 99\%$
$\gamma = 3$	0.0	0.0	0.0	0.0001	0.1174	1.0
$\gamma = 6$	0.0	0.0	0.0001	0.0996	0.9601	1.0
$\gamma = 12$	0.0	0.0	0.0762	0.9555	1.0	1.0
$\gamma = 25$	1.15×10^{-6}	0.3166	0.9430	1.0	1.0	1.0
$\gamma = 50$	0.0052	0.7421	1.0	1.0	1.0	1.0
$\gamma = 100$	0.4783	0.9999	1.0	1.0	1.0	1.0

Table 4.11: Experimental results of the subset attack success rate on the fingerprinting scheme for categorical data, using the Adult data

	$p' = 30\%$	$p' = 60\%$	$p' = 80\%$	$p' = 90\%$	$p' = 95\%$	$p' = 99\%$
$\gamma = 3$	0.0	0.0	0.0	0.004	0.22	1.0
$\gamma = 6$	0.08	0.18	0.20	0.354	0.954	1.0
$\gamma = 12$	0.078	0.0	0.212	0.97	1.0	1.0
$\gamma = 25$	0.012	0.284	0.99	1.0	1.0	1.0
$\gamma = 50$	0.346	1.0	1.0	1.0	1.0	1.0
$\gamma = 100$	0.976	1.0	1.0	1.0	1.0	1.0

of a subset attack on the Adult dataset, however, in latter the rate of success is larger overall. Although the detection algorithm can detect the correct fingerprint from the full set of tuples, the errors introduced by modulo operation are enhancing the success of the attack. Therefore, only for small values of γ, the scheme is resistant to subset attack if the large portion of tuples is not deleted.

4.4 Superset attack

In a superset attack, an attacker adds additional tuples to the fingerprinted data, creating a bigger, pirated dataset. This attack considers only the addition of the new tuples while the original set of tuples remains in the pirated dataset. The sources of the additional tuples can be various. One can add the tuples from other sources such as related datasets with similar attributes, artificial tuples with some semantic meaning, tuples generated from the dataset itself, or the values can be completely random. This attack can only be applied on fingerprinting schemes whose defence algorithms can and do function without the access to the original dataset (e.g. AK Scheme). Otherwise, it is trivial to compare the pirated dataset to the original and remove the tuples that are added by an attacker.

Defence in some cases can be helped by syntactical examination of the dataset. In cases where additional tuples are generated completely randomly, it might be easy for a human to spot and remove them without any algorithm. Also, any semantic background that

the defence knows about the database can serve as the preliminary step in the deletion of additional tuples.

4.4.1 AK Scheme

Assume that fingerprint bit f_i is embedded in the original data ω_i times, and that it is extracted from the additional tuples ω_i' times [1]. Matching of the single extracted fingerprint bit to the correct value f_i is modelled as an independent Bernoulli trial with probability 0.5 of success or failure. (Extraction is controlled by the new unknown values of the primary key, therefore it is equally likely for the extracted bit to be 0 or 1.) For the detection algorithm to fail in extracting the correct fingerprint bit, at least $(1-\tau)(\omega_i+\omega_i')$ embedded bits that correspond to the fingerprint bit f_i must be detected wrongly. Thus, the probability that the fingerprint is detected incorrectly is $B((1-\tau)(\omega_i+\omega_i');\omega_i',0.5)$. The probability that the entire fingerprint is detected incorrectly is

$$fm = 1 - \prod_{i=0}^{L-1}(1 - B((1-\tau)(\omega_i+\omega_i');\omega_i',0.5)) \tag{4.10}$$

Let us set $\tau = 0.5$ (the usual default value) and analyse the performance of superset attack success for that case. Knowing that all fingerprint bits will be correctly extracted from the original tuples because they are not changed, intuitively, in the tuples added by the attacker the incorrect occurrence of some fingerprint bit must outnumber its occurrences in the original tuples. Formally, superset attack can be successful only if $\exists i \in \{0,...,L-1\} : \omega_i' \geq \omega_i$. To make that possible, the attacker must add at least $100\%\eta$ tuples to the original data.

In AK Scheme the choice of fingerprint bit to be embedded is made randomly and independently in each step, therefore we can assume $\omega_i \approx \omega_j, \forall i,j \in \{0,...,L-1\}, i \neq j$. Furthermore, we can also assume $\omega_i' \approx \omega_j', \forall i,j \in \{0,...,L-1\}, i \neq j$. Table 4.12 shows the success of the superset attack depending on γ and number of added tuples. The success is calculated using Equation (4.10), taking into account the assumptions made above. We analyse the success of superset attack for Forest Cover Type data with $L = 96$ and $\tau = 0.5$ Value 0 is the complete failure of the attack and 1 maximum success.

Table 4.12: Superset attack success rate on the Forest Covertype data

ω_i'	$\gamma = 25$	$\gamma = 50$	$\gamma = 100$
$100\%(\omega_i)$	1.35835×10^{-71}	3.61112×10^{-35}	8.32668×10^{-17}
$200\%(\omega_i)$	1.91325×10^{-27}	8.66310×10^{-14}	1.81092×10^{-6}
$300\%(\omega_i)$	8.32595×10^{-18}	9.89990×10^{-9}	0.000439
$400\%(\omega_i)$	3.31239×10^{-13}	1.55168×10^{-6}	0.006221
$500\%(\omega_i)$	1.74078×10^{-10}	0.000047	0.029978
$1000\%(\omega_i)$	0.000045	0.019113	0.536146

The ratio ω_i/ω_i' is approximately the same as ratio η/η', where η' is the number of added tuples because of the randomness in embedding the fingerprint, so we can interpret the percentages in the first row of Table 4.12 as the amount of added tuples in with respect to the original number of tuples.

The success rates in the Table 4.12 are very small even for cases with a big number of added tuples, i.e. the scheme is very resilient to superset attack. Adding a lot of "fake" tuples to the dataset, in this case even several times more than the original dataset size, violates the credibility of the data. The attacker does not benefit from such attack, so the subset attack is usually combined with some other attack such as subset attack (*mix-and-match* attack [1]).

Superset attack requires the generation of new "fake" rows which can be achieved in several ways. The attacker might come up with brand new values and combine them into new tuples or use the existing ones. In any case, the semantic problem might arise if the new values, or a combination of them, do not contextually fit the attribute or the dataset. For example, assume we have a dataset with information about persons such as their age, weight, height, etc. Generating new tuples might result in having tuples with illogical values for attributes, for example *height*:240cm or a same person with values *age*:10y. and *height*:190cm. In this case, the defence against the attack might be trivial, only by removing such tuples from the pirated dataset.

One possible superset attack model consists of generating new tuples from the existing fingerprinted data and mixing those with the original fingerprinted data to create a pirated dataset. The tuples are generated such that every attribute value (except for the private key) is independently randomly chosen from the set of existing values for that attribute. This way we make it harder for a human to distinguish between the tuples originating from the originally fingerprinted data and those added by the attacker, making the defence rely entirely on detection algorithm. The example of tuple generation is shown in Table 4.13. The original tuples from which the values are generated are singled out and the sampled values are highlighted. In the last row, we can see the generated tuple.

4.4.2 Block Scheme

Superset attack essentially does not work if the original dataset is available in the fingerprint detection process. Block Scheme detection algorithm requires the original dataset to extract the fingerprint as discussed in Section 3.5, therefore this kind of attack alone, without a combination with some other attack does not make much sense. The owner trivially removes the tuples from the pirated dataset that are not part of the original and proceeds with the detection algorithm.

Table 4.13: Example of tuple generation

Id	El.	Asp.	Slope	HD-H	VD-H	HD-R	HS-9am	HS-noon	HS-3pm	HD-FP	C-Type
75	2864	118	18	**201**	74	4567	248	221	93	4849	**2**
191	2995	173	15	268	**135**	6312	228	246	146	4135	**2**
893	2924	270	10	134	11	5066	194	244	**189**	1652	5
1258	2803	**57**	32	323	136	342	223	157	45	1851	5
2165	**3390**	59	10	124	12	2610	228	219	124	2357	7
6880	2983	297	**11**	713	-51	1543	190	237	186	**2003**	**2**
11643	2911	332	**11**	67	2	4972	193	225	172	1724	5
19210	2737	20	7	95	14	**2314**	215	225	147	6815	**2**
23117	2877	165	3	30	4	5236	**222**	240	154	4279	1
23135	2801	116	9	30	2	4709	236	**231**	126	4807	**2**
581012	3390	57	11	201	135	2314	222	231	189	2003	2

4.4.3 Fingerprinting scheme for the categorical data

The blind, simplistic fingerprinting scheme described in Section 3.7 behaves in the same way as the AK Scheme against the superset attack. Therefore, we refer to Section 4.4.1.

The second discussed scheme which is based on neighbourhood search is not blind. It means that any additional fake tuples in the dataset would easily be detectable and removed by a simple check and comparison with the original dataset.

4.5 Bit-flipping attack

The subset and superset attacks described in Section 4.3 and Section 4.4 respectively, are targeting the disruption of tuples without changing the values inside the dataset. However, the attacker might change values by selecting some bits and flipping their values in an attempt to destroy the fingerprint. This kind of attack is called a bit-flipping attack. The choice of the bits is random because the attacker is modelled such that he has no knowledge about the owner's secret key that is crucial for fingerprint insertion scheme.

4.5.1 AK Scheme

Let p be the probability that the attacker flips some k^{th} least significant bit, with $p \leq 0.5$ (otherwise fingerprint detection can be applied to transformed data by flipping each fingerprintable bit back). The attacker chooses bits independently, therefore bit flipping is modelled as an independent Bernoulli trial with probability p of success and $1 - p$ of failure. Assume that each fingerprint bit f_i is embedded ω_i times. The detection algorithm will fail to extract the correct fingerprint bit if the correct bit value is extracted less than the defined number of times which is controlled by parameter τ. Therefore at least $(1 - \tau)\omega_i$ embedded bits must be flipped by an attacker. Thus, the probability that

the fingerprint f is detected incorrectly is

$$fm = 1 - \prod_{i=0}^{L-1} (1 - B(\lfloor (1-\tau)\omega_i \rfloor; \omega_i, p)) \tag{4.11}$$

Because the choice of fingerprint bits to be embedded in fingerprint embedding process is completely random, we can assume that each fingerprint bit will be embedded in the data approximately same number of times; $\omega_0 = \omega_1 = ... = \omega_{L-1} = \omega, \forall i, j \in \{1, L-1\}, i \neq j$. Therefore, we can write the expression for false miss rate (Equation (4.11)) as $fm = 1-(1-B(\lfloor (1-\tau)\omega \rfloor; \omega, p))^L$. The results in Table 4.14 and Table 4.15 are obtained using this approximation of false miss rate. Since $0 < B(\lfloor (1-\tau)\omega \rfloor; \omega, p) \leq 1$, by increasing L the false miss rate also increases, meaning that longer fingerprints lead to schemes more vulnerable to bit-flipping attack. Intuitively, longer fingerprint means fewer embeddings of each single fingerprint bit in the data and larger probability for the attacker to erase all of the embeddings of some fingerprint bit, making the detection algorithm incapable of detecting a valid fingerprint.

The effect of parameter γ is shown in Table 4.14 by calculating probabilities of success of bit-flipping attack for different values of γ and p (probability of flipping a bit, i.e. the approximate amount of tuples containing a value with a flipped bit). We choose $\gamma = \{6, 12, 25, 50, 100, 200\}$ and $p = \{45\%, 40\%, 30\%, 20\%\}$, and set $\tau = 0.5$ and $L = 96$ as fixed values. ω is calculated as $\eta/(L * \gamma)$, and $\eta = 581,012$ for the convenience of comparing these results to the experimental results on Forest Covertype dataset.

Table 4.14: Probability of a successful bit-flipping attack on the AK Scheme

	$p = 20\%$	$p = 30\%$	$p = 40\%$	$p = 45\%$
$\gamma = 6$	0	0	6.6530×10^{-9}	0.0671
$\gamma = 12$	0	0	0.0003	0.7327
$\gamma = 25$	0	5.8392×10^{-9}	0.0941	0.9987
$\gamma = 50$	0	0.0002	0.7144	1
$\gamma = 100$	2×10^{-5}	0.0839	0.9994	1
$\gamma = 200$	0.0220	0.8060	1	1

The effect of parameter τ is shown in Table 4.15. The success of the bit-flipping attack is calculated for different values of τ and p, while $L = 96$ and $\gamma = 50$ are set as fixed values.

Both τ and γ affect the success of the attack such that if they are increased, the probability that fingerprint will not be detected correctly is also increased, i.e. the scheme is more susceptible to bit-flipping attack. Generally, if the attacker chooses to flip more bits, i.e. increases p, that also increases his chance for the successful attack, but in the same time violates credibility of the data since more original values would be changed. Note that parameter ξ does not affect the success of the attack because it is not important which LSB of a chosen value is flipped by the attacker.

Table 4.15: Probability of a successful bit-flipping attack on the AK Scheme

	$p = 45\%$	$p = 40\%$	$p = 30\%$	$p = 20\%$
$\tau = 0.50$	1	0.7144	0.0002	0
$\tau = 0.55$	1	0.9999	0.0228	0
$\tau = 0.60$	1	1	0.5779	2×10^{-5}
$\tau = 0.65$	1	1	1	0.0048
$\tau = 0.70$	1	1	1	0.3041
$\tau = 0.75$	1	1	1	0.9996

Misattribution false hit Let us now analyse the probability of detecting valid but incorrect fingerprint caused by a bit-flipping attack - misattribution false hit fh^A. The detection algorithm will extract a binary bit for fingerprinting bit f_i if either at most $\lfloor(1-\tau)\omega_i\rfloor - 1$ or at least $\lfloor\tau\omega_i\rfloor$ of its embedded bits are flipped. If the detection algorithm extracts a binary string, the probability that the binary string is valid but belongs to an innocent buyer is $\frac{N-1}{2^L}$. Therefore, the probability of detecting valid but incorrect fingerprint is

$$fh^A = \frac{N-1}{2^L} \prod_{i=0}^{L-1} (1 - B(\lfloor(1-\tau)\omega_i\rfloor; \omega_i, p) + B(\lfloor\tau\omega_i\rfloor; \omega_i; p)). \tag{4.12}$$

False miss rate is by definition the sum of misattribution false hit and false negative rate. Therefore, false negative is straightforward:

$$\begin{aligned} fn =& 1 - \prod_{i=0}^{L-1} (1 - B(\lfloor(1-\tau)\omega_i\rfloor; \omega_i, p)) - \\ & \frac{N-1}{2^L} \prod_{i=0}^{L-1} (1 - B(\lfloor(1-\tau)\omega_i\rfloor; \omega_i, p) + B(\lfloor\tau\omega_i\rfloor; \omega_i; p)). \end{aligned} \tag{4.13}$$

Experiments We have run experiments on the Forest dataset and obtained results that are shown in table 4.16. As it is shown in the previous section, the success of bit-flipping attack is influenced by multiple parameters - length of the fingerprint L, fingerprint detection assurance parameter τ, amount of marked values γ (or equivalently, number of embedded bits corresponding to a single bit ω) and the attacker's parameter p that defines the number of flipped bits in pirated data. For our experiments we fix the values of $L = 96$ and $\tau = 0.5$. We run 100 experiments with different random bit-flipping pattern for each of the combinations of parameters $\gamma = 6, 12, 25, 50, 100$ and $p = 20\%, 30\%, 40\%, 45\%$.

The experimental results show that flipping 40% of the bits available for fingerprinting most likely deletes the fingerprint. At 30% the schemes with bigger γ ($\gamma = 50, 100$), i.e. less embedded marks, fail under the bit-flipping attack. Generally, the scheme is more

Table 4.16: Experimental results of a bit-flipping attack on the AK Scheme, using the Forest Cover Type data

	$p = 20\%$	$p = 30\%$	$p = 40\%$	$p = 45\%$
$\gamma = 6$	0	0	0.50	0.56
$\gamma = 12$	0	0	0.50	1.0
$\gamma = 25$	0	0	0.54	1.0
$\gamma = 50$	0	0.50	0.72	1.0
$\gamma = 100$	0	0.86	1.0	1.0

robust against the attack for a smaller value of γ. The empirical results differ from the analytic results in Table 4.14. The two analysis agree on the cases where the attack completely succeeds and completely fails, however, in our experiments the scheme appears less robust to the bit-flipping attack than it is suggested in the analysis. The success rates in Table 4.14 are calculated using Equation (4.11) under the assumption that all fingerprint bits are embedded the same number of times, which is approximated with ω. This is not true due to the random nature of bit choice. In reality, every fingerprint bit has it's own ω_i value. Many fingerprint bits, thus, are embedded in data less than ω times and are easier to destroy. Since destroying all occurrences of just one fingerprint bit makes the fingerprint impossible to extract, the low occurrences of some fingerprint bits might decrease the robustness. This might be the reason for the difference in the attack success rates. Even though the probabilities for attack success are different, the boundary for robustness is confirmed. For $\gamma = \{6, 12, 25\}$ it is safe to flip up to 30% of LSBs to not remove the fingerprint, and for larger values, $\gamma = \{50, 100\}$, it is safe to flip around 20% of the LSBs.

4.5.2 Block Scheme

Assume that the attacker examines every bit available for fingerprinting independently and selects it for flipping with probability p. Let us approximate the number of times that each fingerprint bit is embedded in the data to ω. For the detection algorithm to fail to recover the correct fingerprint bit, at least $(1 - \tau)\omega$ embedded bits corresponding to the single fingerprint bit f_i must be changed, i.e. more than $\omega - \lceil \tau\omega \rceil + 1$ bits must be changed. Probability that one fingerprint bit is destroyed is $B(\omega - \lceil \tau\omega \rceil + 1; \omega, p)$. The probability that the entire fingerprint will be detected incorrectly is therefore

$$fm = 1 - (1 - B(\omega - \lceil \tau\omega \rceil + 1; \omega, p))^L \tag{4.14}$$

Table 4.17 shows the probabilities that the bit-flipping attack will be successful on Block Scheme, depending on p and β. Parameters are set as follows: $\xi = 2$, $\tau = 0.5$ and $L = 96$. We choose $\eta = 581012$ and $v = 10$, same as Forest Covertype dataset.

Experiments We run experiments on the Forest dataset with the previously defined parameters. Table 4.18 shows the obtained empirical results for the success of the

Table 4.17: Probability of a successful bit-flipping attack on the Block Scheme

	p=30%	p=40%	p=45%	p=50%
$\beta = 5$	0	0	0	1.0
$\beta = 10$	0	0	0.02	1.0
$\beta = 15$	0	0.002	0.88	1.0
$\beta = 20$	0	0.017	0.97	1.0

bit-flipping attack on the Block Scheme. Each experiment is run 100 times and the table shows the average success of the bit-flipping attack.

Table 4.18: Experimental results of the bit-flipping attack on the Block Scheme, for the Forest Cover Type data

	p=30%	p=40%	p=45%	p=50%
$\beta = 5$	0	0	0.50	1.0
$\beta = 10$	0	0.50	0.50	1.0
$\beta = 15$	0	0.50	0.92	1.0
$\beta = 20$	0.08	0.50	1.0	1.0

The experiments confirm the rule that having more marks in the data, i.e. smaller β, makes the scheme more robust against the bit-flipping attack. Our experimental rates of the attack success, however, differ from the analytical results in Table 4.17. The change might be due to the implementation limitations introduced by the design of the Block Scheme. We address in Section 3.5.2 the problem of "extra data" that in reality never gets fingerprinted. This means that less data is fingerprinted, i.e. fewer marks are in the data due to this limitation than it is assumed for the calculation of the theoretical attack success rates. This might be the reason why experimentally this scheme appears to be more vulnerable to the attack than expected by the theoretical analysis.

The scheme with all of the chosen parameters guarantees robustness for to 30% flipped LSBs. Choosing a small value of β, e.g. 5, the tolerated amount of flipped bits rises up to 45%. We can argue that this is indeed a robust scheme. The assumed attacker would like to keep the data useful, and flipping more bits significantly changes the values in the data and the utility decreases.

4.5.3 Two-level Fingerprinting Scheme

In this section, we empirically analyse the robustness of the Two-level Fingerprinting Scheme against the bit-flipping attack. We measure the success on two levels; the ownership verification and the fingerprint extraction. The attacker might be able to either (i) disable both ownership verification and fingerprint extraction, (ii) destroy the fingerprint but fail in disabling ownership verification, or (iii) fail in both. We use the Forest Covertype data for the experiments and measure success of the subset attack over

500 runs of each parameter setting. We choose: $L = 96$, $\xi = 2$, $\alpha_1 = \alpha_2 = \alpha_3 = 0.01$, $\gamma_1 = \gamma_2 = \{10, 25, 50, 100\}$. Figure 4.3 shows the results of the experiments.

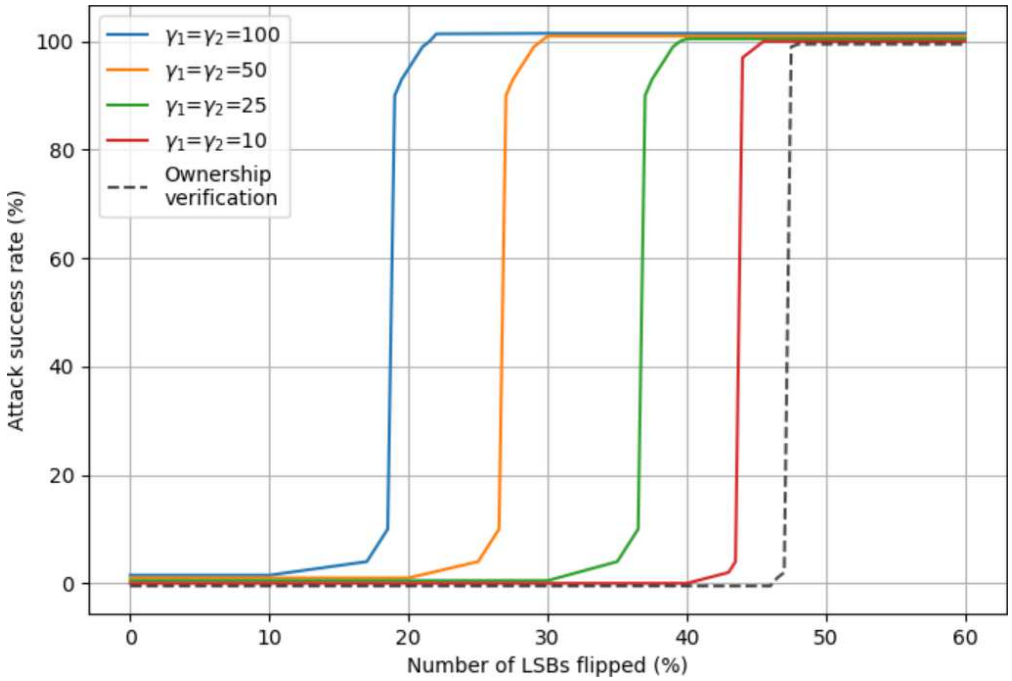


Figure 4.3: Bit-flipping attack success in the Two-level Fingerprinting Scheme

The coloured lines show the success rate of the attacks against the fingerprint extraction. The dashed line shows the success rate of the attack against the ownership verification. The experiments show that the robustness against the bit-flipping attack, as expected, grows with more embedded marks. The scheme with $\gamma_1 = \gamma_2 = 100$ fails to extract the fingerprint when 19% of the LSBs are flipped by the attacker, while the scheme with $\gamma_1 = \gamma_2 = 10$ fails at approximately 43% flipped LSBs. In both cases, the ownership is verified 100% of the times, even though the exact fingerprint cannot be extracted. The ownership verification fails with approximately 47% flipped LSBs. The robustness of the ownership verification is very similar for each γ_1 value, therefore we represent it with only one (dashed black) line in Figure 4.3.

Choosing smaller γ values makes the Two-level Fingerprinting Scheme very robust against the bit-flipping attack. The two-level fingerprint provides the additional protection measure of verifying the ownership even in cases when the exact fingerprint cannot be extracted. Smaller values of γ, however, introduce more error in data. How these errors affect the utility of the data, we discuss in Chapter 5.

4.6 Additive Attack

Consider a scenario where the attacker tries to claim the ownership of the dataset by inserting an additional fingerprint in the bought dataset. We call this strategy an additive attack [35]. The competing ownership claims can be resolved if there exists at least one bit that both the owner and the attacker have marked, each with a different value. The way to resolve the ownership claim competition is to determine which owner's marks win, i.e. which mark has overwritten the other. The winning owner's mark is clearly inserted later, therefore his claim of ownership is false.

Method for dealing with the false claims of ownership could be to ask both the owner and the attacker to produce the original dataset before it was fingerprinted and to demonstrate the presence of the fingerprint in each other's original datasets. The owner will be able to demonstrate the presence of her fingerprint in the attacker's original unlike the attacker in the owner's original.

4.6.1 AK Scheme

In the AK Scheme, it is justified to conclude that the odds of finding such conflicting bits are low. Suppose that the data fingerprinted by the owner is marked ω times with parameters γ, v and ξ and that the attacker performs the fingerprinting insertion algorithm with parameters γ', v' and ξ'. Under the usual probabilistic model of AK Scheme's bit-marking process, the probability that a specified bit marked by original fingerprint is also marked by the attacker is the product of probabilities that the tuple containing the bit is chosen for marking $(1/\gamma')$, that the attribute containing the bit is also chosen for marking $(1/v')$ and that the specified bit is chosen $(1/\xi')$. The probability that the attacker's mark is different from the original mark is $1/2$ so that the overall probability that the specified bit is a conflict bit is $1/(2\gamma' v' \xi')$. The tuples are marked independently of each other, therefore the probability that the attack is successful, i.e. no conflicting bits are found, is

$$P\{success|\omega\} = (1 - \frac{1}{2\gamma' v' \xi'})^{\omega} \tag{4.15}$$

For example, let the dataset have around 500,000 tuples and $\omega = 1000$. Assume that attacker wants to increase his chances of success. If the attacker sets $\gamma' = 10,000$ (a rather big value considering that it means that only 1/10,000 tuples will be marked), $v' = 10$ and $\xi' = 5$, then $P\{success|\omega\} = (1 - 10^{-6})^{1}000 \approx 0.999$.

4.6.2 Block Scheme

The solution from section 4.6.1 is applicable to the block fingerprinting scheme as well. Suppose that the attacker runs the fingerprint insertion algorithm with parameters β', ξ' and v'. Let $1/\gamma'$ be the percentage of tuples marked by the attacker. Due to the uniform distribution of the marks in the Block Scheme, we can approximate the percentage

$1/\gamma \approx (\xi v)/\beta^2$, assuming that there is in average no more than 1 mark in a single tuple. Let the data be marked $L\omega$ times in total by the owner. The probability that the additive attack is successful is then

$$\begin{aligned} P\{success|L\omega\} &= (1 - \frac{1}{2\gamma' v' \xi'})^{L\omega} \\ &= (1 - \frac{1}{2\frac{\beta'^2}{\xi' v'} v' \xi'})^{L\omega} \\ &= (1 - \frac{1}{2\beta'^2})^{L\omega} \end{aligned} \tag{4.16}$$

The success of the additive attack depends exponentially on $L\omega$. The attacker can increase his chances for success by increasing β', however with $L\omega \gg \beta'$, the chances for the successful attack are low. For example, with $\beta' = 30$ and $L\omega = 10000$, $P\{success|L\omega\} = 0.0039$.

4.7 Collusion Attack

Fingerprinting produces distinct copies of the data for each of the buyers. This opens the possibility for multiple buyers to have access to each other's copies of the data, all fingerprinted with different fingerprints, and work in coalition in order to create a useful data copy that would not implicate any member of the coalition. One possibility for the members of the coalition is to attempt to erase the fingerprint or to modify values such that detection algorithm implies an innocent buyer who is not a member of the coalition to be the traitor. Collusion attack is specific for fingerprinting, as opposed to other attacks discussed in this thesis that can be applied in the context of watermarking. Collusion is studied extensively in the literature [74, 75, 76, 77, 78] and collusion resistant fingerprinting codes have been proposed such as one by Blakley et al. [80], by Guth and Pfitzmann [80], and probably the most well-know by Boneh and Shaw - *BoSh* [74].

Boneh and Shaw [74] provide the definitions of collusion-secure codes and propose the methods for construction of codes and algorithm for dealing with collusion attacks. The method can be used for fingerprinting any sort of digital data: documents, multimedia, software, etc. The effectiveness of the approach is based on the *Marking Assumption* that states that "the main property of the marks should be satisfied are that users cannot change the state of an undetected mark without rendering the object useless" [74]. The colluding buyers can detect only the fingerprint bits in which their copies differ, otherwise, the fingerprint cannot be detected. For instance, two buyers with their fingerprinted dataset can fairly easily compare their datasets and remove or change the values that differ between the copies.

The following definitions of collusion-secure codes are defined [74]:

- **c-frameproof code** satisfy that no coalition of at most c members can frame a user who is not part of the coalition. c-frameproof codes prevent this harder version of treason but do not assure that the traitor(s) will be found
- **totally c-secure code** is a code for which exist an algorithm that outputs a member of a coalition with at most c members that generated an arbitrary code under the coalition. It is proven that for $c \geq 2$ the totally c-secure code does not exist.
- **c-secure code with ϵ-error** is a relaxation of the previous - if the code is c-secure with ϵ-error, then there exists an algorithm which outputs a member of coalition with at most c members with probability $1 - \epsilon$, $\epsilon \in [0, 1]$.

BoSh codes are designed to be c-secure with ϵ-error. Increasing c or reducing ϵ provides better security definition, but results in longer codes. BoSh code is generated as a concatenation of b code-words of size $(a-1)d$ from public "inner code" that is common for all buyers. Therefore, the code has length $b(a-1)d$. Code words from the inner code are chosen according to the secret buyer-specific "outer code" and randomly permuted before the use. a, b and d are code construction parameters (in [74] n, L and d respectively, but we change the notation due to overlapping with notation in this thesis). The parameter values are: $a = 2c$, $b = 2c\log(2N/\epsilon)$ and $d = 2a^2\log(4ab/\epsilon)$. It is required that the random permutation and random outer code for each buyer are kept secret from all buyers. The tracing algorithm takes as input pirated data under collusion attack and returns exactly one malicious buyer.

AK Scheme AK insertion algorithm (Algorithm 3.1) is not secure against collusion attack. Each fingerprint bit f_i is embedded to the same place in every fingerprinted copy of the dataset since it only depends on parameters γ, v and ξ which are not buyer-specific, and owner's secret key $\mathcal{K}$. Assume that 3 buyers are in collusion and they extract following marks from the bits that differ within their copies: (1,0,1), (1,1,0) and (0,1,1). They decide to change values according to the majority, so they come up with the new mark (1,1,1) that they use to create the pirated data. Detection algorithm cannot match the new fingerprint to any of the members of the coalition. Another option for members of the collusion is to produce a random mark, e.g. (0,0,1) which leads to the same outcome.

To create a collusion-resistant scheme, authors in [1] propose the modified version of BoSh code that is adequate to be incorporated into AK Scheme. The fingerprint in the proposed solution consists of two parts: (1) watermark which is the same for every buyer and computed from a hash function using owner's secret key, $\mathcal{H}(\mathcal{K})$, and (2) BoSh code. Those two parts are concatenated to create the fingerprint.

There are two main differences between BoSh code and proposed solution [1]:

1. Collusion-resistant AK Scheme does not require recording of secret outer code for every buyer as it is the case for BoSh codes. Storing any kind of secret buyer-specific

information is already discussed to violate key-based property and as a reason for incorporating pseudo-random sequence generator in the context of a fingerprint. The outer code needs to be hidden from the buyers, but deterministic to the owner. Therefore, the outer code is in collusion-resistant AK Scheme generated using such pseudo-random sequence generator from the owner's secret key $\mathcal{K}$ and buyer's ID number.

2. Similarly, the random permutation for all buyers needs to be stored in BoSh code setting. The purpose of the final random permutation in BoSh code is to hide which mark in dataset corresponds to which fingerprint bit. In AK Scheme this permutation is not necessary as the choice of fingerprint bit to be embedded is already random by the design of the insertion algorithm (lines 7 and 8 in Algorithm 3.1).

Insertion and detection algorithms of collusion-resistant AK Scheme are slightly modified insertion (Algorithm 3.1) and detection (Algorithm 3.2) algorithms of AK. The modifications in the insertion algorithm are as follows:

1. Additional parameter L_1 represents the length of the first (watermarking) part of a fingerprint. $L = L_1 + L_2$ (L_2 is the length of second part - BoSh code).
2. Generation of fingerprint replaced by concatenation of watermark part of the fingerprint and BoSh code.
3. Additional parameter c - maximum coalition size
4. Additional parameter ϵ - maximum false detection rate in tracing a coalition

The detection algorithm is modified such that it runs in two consecutive phases:

1. **Watermark check** - in this part, the watermark part $\mathcal{F}_\infty$ of recovered fingerprint template $\mathcal{F}$ is checked against the inserted codeword. The algorithm returns *none suspected* for a single bit mismatch. This phase serves as prevention from 100% misdiagnosis false hit rate in cases where non-pirated data is the input of detection algorithm (because the following phase returns exactly one malicious buyer).
2. **BoSh tracing algorithm** (from [74]) - if algorithm passes the first phase, this phase identifies exactly one malicious buyer.

Authors of [1] analyse the robustness rates of collusion-resistant AK Scheme and carry out the experimental results on scheme robustness. Following the proof from [74] that probability of BoSh tracing algorithm returning the buyer who is indeed a member of the coalition is greater than $1 - \epsilon$, the false miss rate fm and misattribution false hit fh^A satisfy

$$fm = fh^A \leq \epsilon \tag{4.17}$$

The misdiagnosis false hit fh^D when the detection algorithm is applied on unmarked dataset is

$$fh^D = \prod_{i=0}^{L_1-1} B(\lfloor \tau\omega_i \rfloor; \omega_i, 0.5) \leq \frac{1}{2^{L_1}}) \tag{4.18}$$

and it can be decreased exponentially by increasing L_1 ($fh^D \simeq 0$ if $L_1 \gg 1$).

4.8 Summary

In this chapter, we analysed the robustness of the fingerprinting techniques under certain types of attacks. We address subset attack, superset attack, bit-flipping attack, additive attack and collusion attack. As opposed to analysing the inability of a scheme to detect the fingerprint under a malicious attack, we also address the possibility of detecting the fingerprint in unmarked data - misdiagnosis false hit.

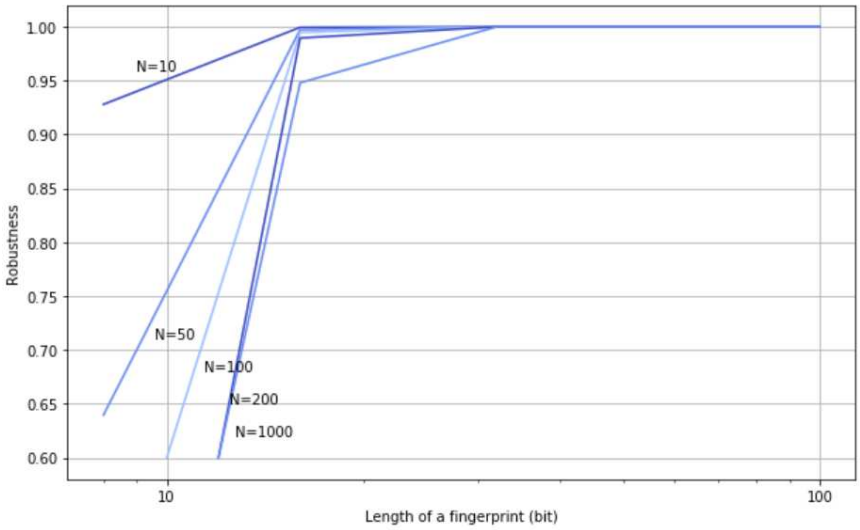


Figure 4.4: Misdiagnosis false hit rate

To avoid the misdiagnosis false hit, the fingerprint length must be big enough. Figure 4.4 shows the robustness of a scheme given the assumed number of buyers. We see that for the larger number of buyers we must ensure the longer fingerprint, e.g. if there are 100 buyers, the fingerprint length L should be larger than 30.

The additive attack is, like misdiagnosis false hit, not specific for a certain scheme. The robustness of the scheme against the additive attack depends on the number of marks the scheme embeds in the data, which is controlled by parameters γ in AK Scheme and the Scheme for fingerprinting categorical data, by γ_1 and γ_2 in Two-level Scheme and by β in the Block Scheme. We show in Figure 4.5 the relation between number of marks and robustness of the scheme for different attacker's embedding patterns. The attacker's parameter of the ratio of the marked tuples is γ' ($\gamma' = 1,000$ means that one

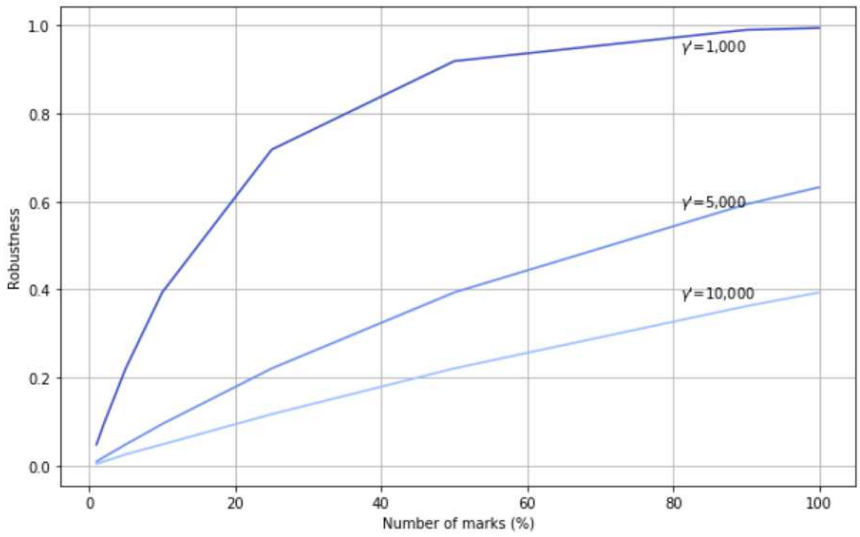


Figure 4.5: Robustness against the additive attack

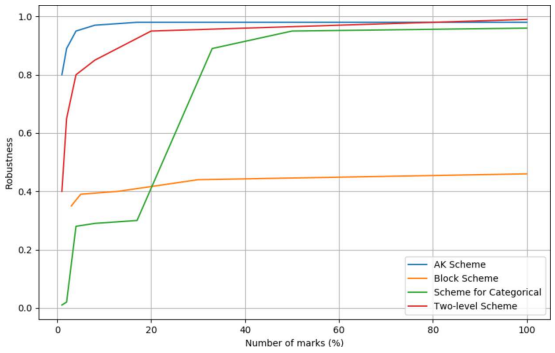


Figure 4.6: Comparison of robustness against the subset attack

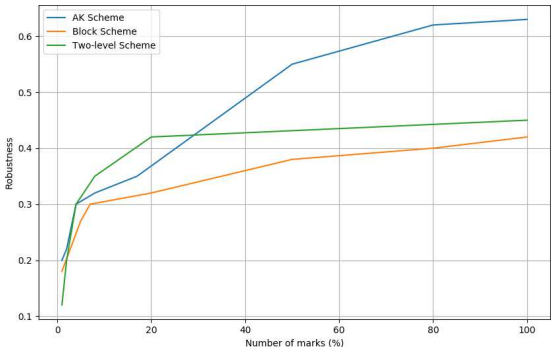


Figure 4.7: Comparison of robustness against the bit-flipping attack

in a thousand tuples is marked). It depends on the attacker how he chooses to embed his fake fingerprint. If the dataset is very big and marking one in 10,000 tuples is sufficient to claim his ownership, then the fingerprinting schemes are not very robust - there is only 40% chance that the attacker will not succeed. However, smaller datasets (10,000 tuples or less) might be harder to attack since the attacker would need to choose smaller γ' to be able to claim the ownership. Schemes are much more robust if the attacker chooses $\gamma' < 1,000$.

Furthermore, we discussed the robustness against the attacks where the attacker modifies the dataset, e.g. subset attack and bit-flipping attack. A general conclusion is that more marks in the data mean better robustness. A somewhat smaller role plays the parameter ξ that defines how many LSBs are available for fingerprinting such that is ξ is bigger,

the scheme is more robust. We compare the robustness of the discussed fingerprinting techniques against the subset attack in Figure 4.6 and the bit-flipping attack Figure 4.7. Robustness against the subset attack is measured as the largest percentage of tuples deleted where the scheme still detects the correct fingerprint with 100% probability. The AK Scheme shows the best robustness even for a smaller number of marks in the data. The Two-level Scheme and the scheme for categorical data are reaching a high robustness level for a larger number of marks in the data. The Block Scheme under-performs all the schemes.

Robustness of the bit-flipping attack is measured as the largest percentage of the LSBs that can be flipped while the scheme would still detect the correct fingerprint with probability 100%. AK Scheme again outperforms other schemes, except for the smaller number of marks where the Two-level Scheme is more robust. The Block Scheme is the least robust scheme against the bit-flipping attack. However, we can argue that all of the schemes are robust against the attack since they can all reach the level of robustness of > 0.4.

We conclude that a good robustness level can be reached with a sufficient number of marks in the data. However, more marks in the data introduce more errors and they might significantly affect the utility of the data. In the next chapter, we will analyse how the utility is affected and if introducing the necessary number of marks to ensure the robustness is acceptable from the utility point of view.

CHAPTER 5

An evaluation on the utility of the fingerprinting schemes

5.1 Quality effects on fingerprinted datasets

To get an insight into how the fingerprinting scheme affects the quality of the data that is disturbed by perturbations, we calculate the mean and variance of the values of an attribute. In this section, we first present the analysis and the empirical evaluation of changes in these two measures for each of the fingerprinting schemes described in Chapter 3. Exclusively, for the fingerprinting scheme for categorical data, we use a different quality measurement since the mean and variance do not apply to the categorical values. Instead, we count the number of changes in the data introduced by fingerprinting.

5.1.1 AK Scheme

The procedure of embedding the fingerprint is determined by the owner's secret key $\mathcal{K}$, primary key attribute P and controlled by parameters γ, v and ξ. In a dataset with η tuples, on average η/γ tuples are selected for marking. In a selected tuple a single bit of a single attribute will be selected for marking. Mark value is calculated as a result of applying XOR function on fingerprint bit and pseudorandomly selected mask bit. The mark bit value will half of the times match the original value, on average, and therefore leave the selected bit unchanged. Considering that the probability of selecting a value for marking is $1/(\gamma v)$, i.e. one over the number of selected tuples times number of attributes, and the probability that the mark bit differs from the original bit value is $1/2$, we obtain the probability that the value will be selected and changed

$$P\{L_i = 1\} = 1 - P\{L_i = 0\} = \frac{1}{2\gamma v} \tag{5.1}$$

where L_i equals 1 if the value of a certain attribute of tuple i is selected and changed (in contrast, it equals 0 if the new value doesn't differ from the original value).

In addition, we assume that the original values of an attribute in the dataset are $x_1, x_2, ..., x_\eta$ and the values of the attribute after fingerprinting are $x_1+\Delta_1, x_2+\Delta_2, ..., x_\eta+\Delta_\eta$. $\{\Delta_1, \Delta_2, ..., \Delta_\eta\}$ represent the error caused by fingerprinting. They are independent and identically distributed random variables. We further assume the representation of $\Delta_i, 1 \leq i \leq \eta$,

$$\Delta_i = L_i S_i 2^{U_i} \tag{5.2}$$

where $S_i \in \{-1, 1\}$ depending on whether the perturbed value is smaller or greater than the original, both with probability 0.5, and $U_i \in \{0, 1, ..., \xi - 1\}$ is uniformly distributed variable representing position of the marked bit.

Mean The mean value of the original attribute values is

$$\overline{x} = (1/\eta) \sum_{i=1}^{\eta} x_i \tag{5.3}$$

and the mean of the attribute values after embedding the fingerprint is

$$\overline{x}' = \overline{x} + \overline{\Delta} \tag{5.4}$$

where

$$\overline{\Delta} = (1/\eta) \sum_{i=1}^{\eta} \Delta_i \tag{5.5}$$

The expected error of a single attribute value is

$$E[\Delta_i] = \frac{1}{2} L_i 2^{U_i} - \frac{1}{2} L_i 2^{U_i} = 0, \forall i : 1 \leq i \leq \eta \tag{5.6}$$

thus the expected error in attribute mean value after embedding the fingerprint is

$$E[\overline{\Delta}] = 0 \tag{5.7}$$

Variance The variance of the original attribute values is given by

$$V_x = \frac{1}{\eta}\sum_{i=1}^{\eta}(x_i - \overline{x})^2 \tag{5.8}$$

and the variance of the perturbed attribute values after the fingerprint insertion is

$$V_{x+\Delta} = \frac{1}{\eta}\sum_{i=1}^{\eta}[(x_i + \Delta_i) - (\overline{x} + \overline{\Delta})]^2 \tag{5.9}$$

After applying some algebra, the error in variance is given by

$$V_{x+\Delta} - V_x = \frac{1}{\eta}\sum_{i=1}^{\eta}(\Delta_i - \overline{\Delta})^2 + 2 \cdot \frac{1}{\eta}\sum_{i=1}^{\eta}(x_i - \overline{x})(\Delta_i - \overline{\Delta}) \tag{5.10}$$

The expected error in computing the variance is given by

$$E[V_\Delta] \approx \frac{2^{2\xi}}{6\gamma v \xi} \tag{5.11}$$

Table 5.1: Change in variance introduced by fingerprinting with AK Scheme

		γ	100		50		25		12	
		ξ	4	8	4	8	4	8	4	8
	Expected error in variance		0	1.4	0.01	2.7	0.02	5.5	0.04	11.4
Attribute	**Mean**	**Variance**								
Elevation	2,959	78,391	0	+1	0	+1	+1	+5	+1	+9
Aspect	156	12,525	0	+1	0	+1	+1	+5	0	+8
Slope	14	56	0	+1	0	+3	0	+5	0	+11
HD-Hydrology	269	45,177	0	+1	0	+1	0	+2	+1	+2
VD-Hydrology	46	3,398	0	+1	0	+2	0	+4	0	+9
HD-Roadways	2,350	2,431,276	0	+10	0	+10	-1	+5	+2	+37
Hillshade-9am	212	717	0	+1	0	+2	0	+4	0	+9
Hillshade-noon	223	391	0	+1	0	+2	0	+4	0	+10
Hillshade-3pm	143	1,465	0	+1	0	+2	0	+4	0	+8
HD-Fire-Points	1,980	1,753,493	0	-2	0	+5	0	+8	+1	+30

Experiments Experimental results are obtained by embedding a fingerprint into the Forest dataset. We choose the set of following values for parameter $\gamma = \{12,25,50,100\}$, and $\xi = \{4,8\}$. Table 5.1 contains recorded changes in the variance introduced by fingerprinting for each of the attributes and parameter setting. The results support the analysis previously made on errors in the mean and variance of the attribute values. The error in the mean in all of the cases of this experiment was < 0.01, so only the error in

the variance is presented. The largest changes are expectedly occurring when γ is small and ξ is big, i.e. when more tuples are selected and more bits of the value are available for marking. The errors in variance between cases with the same γ value and different ξ differ significantly, implying that imperceptibility of the fingerprint is highly sensitive to the number of LSBs available for marking. Original values of the variances, in general, do not affect the relative error of perturbed values. In rare cases, the introduced errors result in a decrease of the variance, however, the change is in the same range as in cases where the variance increases.

5.1.2 Block Scheme

Experimental results are obtained by embedding a fingerprint into the Forest dataset. We choose the set of following values for parameter $\beta = \{30,25,15,10\}$, and $\xi = \{4,8\}$. Table Table 5.2 contains recorded changes in mean introduced by fingerprinting for each of the attributes and parameter setting, and Table 5.3 changes in variance.

Table 5.2: Change in mean introduced by fingerprinting with the Block Scheme

	β	30		25		15		10	
	ξ	4	8	4	8	4	8	4	8
Attribute	**Mean**								
Elevation	2959								
Aspect	156								
Slope	14						+1		
HD-Hydrology	269								
VD-Hydrology	46				+1		+1		+1
HD-Roadways	2350								
Hillshade-9am	212								
Hillshade-noon	223								-2
Hillshade-3pm	143				-1		-1		-1
HD-Fire-Points	1980								

The error is primarily controlled by parameter β. We have previously discussed the expected average number of changes which is described with Equation (3.5). β in denominator suggests that for smaller β there will be more errors in the data introduced by the fingerprint. We can see that a larger number of errors expectedly introduces larger errors in variance from the Table 5.3. The smallest error of variance appears for $\beta = 30$ and increases for smaller β values.

A large difference in the error of variance is noticeable between different values of ξ which defines the number of LSBs available for fingerprinting. We expect larger errors if more LSBs are available for fingerprinting - bits of more significance are being potentially changed, therefore creating bigger distortions (also assumed from Equation (3.5)). The experiment results in Table 5.4 and Table 5.2 confirm this claim. Setting $\xi = 4$ will result

Table 5.3: Change in variance introduced by fingerprinting with the Block Scheme

	β	30		25		15		10	
	ξ	4	8	4	8	4	8	4	8
Attribute	**Variance**								
Elevation	78391	0	+13	+1	+15	+1	+48	+1	+178
Aspect	12525	0	+7	0	+12	0	+35	0	+127
Slope	56	0	+12	0	+18	0	+48	0	0
HD-Hydrology	45177	0	+6	+1	+4	+1	+13	+2	0
VD-Hydrology	3398	0	+10	0	+15	0	+38	0	+87
HD-Roadways	2431276	0	+3	0	+3	0	+44	-2	0
Hillshade-9am	717	0	+11	0	+15	0	+41	0	+8
Hillshade-noon	391	0	+11	0	+16	0	+45	0	+200
Hillshade-3pm	1465	0	0	0	+13	0	+35	0	+160
HD-Fire-Points	1753493	0	0	0	-4	0	+54	0	+68

in no or very small errors of variance and mean. Note that $\xi = 8$ may change a value in the dataset up to $\pm 2^7 = \pm 128$. Considering that most of the values in Forest dataset are in the range of 2^7, $\xi = 8$ is a rather big value. For example, in the attribute *Slope* where the mean value is 14 (see Table 5.2), this may cause quite a perceptible modification. For attributes with a range of larger values, e.g. *Aspect* with mean 156 or *HD-Roadways* with mean 2350, the error of $\leq \pm 128$ causes slightly less perceptible modifications in single values compared to *Slope*, however the overall variance is certainly affected. We see in Table 5.2 that error in the mean is introduced only for $\xi = 8$ and for attributes with the small mean value. We have set $\xi = 8$ for experiments purely to show the clear distinction of the ways that different values of ξ affect the quality of the data. In practice, ξ is set according to the values in the dataset and generally to the smaller values. When ξ is set appropriately, in these experiments $\xi = 4$, different values of β do not affect the errors in mean and variance too much.

5.1.3 Two-level Fingerprinting Scheme

The embedding process is controlled by γ_1, γ_2 and ξ. Using Equation (3.6), a dataset with η tuples will have $\frac{1}{\gamma} = \frac{1}{\gamma_1} + (1 - \frac{1}{\gamma_1}) * \frac{1}{\gamma_2}$ tuples selected for marking. Due to the pseudo-random nature of bit marking, the selected bit's original value will half of the times match the mark. Thus, the probability that the value will be selected and changed is

$$P\{L_i = 1\} = 1 - P\{L_i = 0\} = \frac{1}{2\gamma} = \frac{1}{2\gamma_1} + (1 - \frac{1}{\gamma_1}) * \frac{1}{2\gamma_2} \tag{5.12}$$

where L_i equals 1 if the value of a certain attribute of tuple i is selected and changed (in contrast, it equals 0 if the new value doesn't differ from the original value).

The authors of the proposed scheme in their work [49] present the quality effects by measuring mean and variance before and after embedding. They show the results from experiments run on a portion of Forest Cover data ($\eta = 5,000$) and where fingerprinting is applied to a single attribute - *Elevation*. Both mean and variance showed minor absolute alteration (≈ 0.01) and lead to the conclusion that the scheme preserves the utility of the data.

We show the empirical results on change in mean and variance in a scenario with entire Forest Cover Type data ($\eta = 581,012$). For our experiments we consider all 10 numerical attributes for marking ($v = 10$) and set parameters as follows: $\gamma_1 = \gamma_2 = \{10, 25, 50, 100\}$, $\eta = \{4, 8\}$ and $L = 96$.

Table 5.4: Change in variance introduced by fingerprinting using the Two-level Fingerprinting Scheme

		$\gamma_1 = \gamma_2$	100		50		25		10	
		ξ	4	8	4	8	4	8	4	8
Attribute	**Mean**	**Variance**								
Elevation	2959	78391	+1	+3	+1	+4	+1	+4	+1	+14
Aspect	156	12525	0	+3	0	+7	0	+15	0	+36
Slope	14	56	0	+1	0	+2	0	+4	0	+10
HD-Hydrology	269	45177	0	+3	0	+5	+1	+7	+2	+10
VD-Hydrology	46	3398	0	+1	0	+1	0	+3	0	+6
HD-Roadways	2350	2431276	-1	+12	0	+27	0	+31	+1	+61
Hillshade-9am	212	717	0	0	0	+1	0	+2	0	+6
Hillshade-noon	223	391	0	+4	0	+9	0	+18	0	+43
Hillshade-3pm	143	1465	0	+1	0	+3	0	+5	0	+13
HD-Fire-Points	1980	1753493	+1	0	+1	+12	0	+19	+1	+34

The total mark ratio in the embedding process is around $1/\gamma_1 + 1/\gamma_2$. Table 5.4 shows that the change in variance increases for larger number of marks, i.e. smaller γ_1 and γ_2. The change, however, depends on the number of LSBs available for marking ξ as well. By choosing a smaller value, the change of variance can be almost completely avoided. The smaller ξ value, however, decreases the robustness of the scheme which needs to be taken into account when choosing the value of this parameter. This is discussed in Chapter 4. The mean value of any attribute did not change more than 0.01 in any of the experiments. These changes are minuscule enough to hold the claim that the alteration for fingerprinting does not affect the usability of the data.

5.1.4 Fingerprinting scheme for categorical data

The fingerprinting scheme that deals with categorical data requires a different type of measure for data utility since mean and variance are not applicable in this case. One possible measure is the number of changes introduced by marking the data.

We fingerprint the categorical values of the Adult dataset using the fingerprinting scheme for categorical data. Furthermore, we fingerprint the numerical attributes using the AK Scheme. Table 5.5 shows the utility effects on the Adult dataset (which contains 30,162 tuples) introduced by the fingerprinting scheme for categorical data. The utility of numerical attributes is still measured by mean and variance, where the difference in the mean is negligible (it does not exceed 0.02 and is therefore excluded from the table). The change in variance introduced by errors for numerical attributes is also rather small, as it was the case with previously presented schemes. For each categorical attribute, we count how many changes in values are introduced by the fingerprint. The Number of values that change in a single categorical attribute is approximately $30,162/(2\gamma v)$. For the presented set of parameters, the introduced total number of changes is $< 4\%$ of the total number of tuples in the dataset. Due to the random nature of fingerprint insertion process, the distributions of attributes are not significantly affected.

Table 5.5: Change in variance and value-flips introduced by fingerprinting with the fingerprinting scheme for categorical data and the AK Scheme, on the Adult dataset

	γ	50		25		12		6	
	ξ	2	4	2	4	2	4	2	4
Attribute	**Variance**								
Age	173	0	0	0	0	0	0	0	+0.05
Capital Gain	54,853,968	-1	-3	-5	-11	-23	-56	-31	-67
Capital Loss	163,457	0	-1	0	-1	-1	-2	-2	-5
Hours per Week	144	0	0	0	0	0	+0.2	0	+0.3
		Value Changes							
Workclass		26	19	45	45	81	90	165	165
Education		26	18	49	43	83	84	172	173
Marital Status		24	24	46	44	101	87	207	189
Occupation		23	20	44	47	75	73	148	135
Relationship		22	22	29	41	81	89	175	189
Race		19	20	47	51	87	91	160	174
Sex		12	5	19	13	39	25	77	46
Native country		19	21	45	30	94	78	173	164

5.2 Impact of fingerprinting on Machine Learning models

In this section, we evaluate the utility of fingerprinted datasets by measuring the difference in the performance of classifiers for a predictive task, using the original and the fingerprinted dataset. We use *accuracy* and *F1* as performance measures. In Machine Learning, binary classification is the task of classifying the elements of a given set into two groups (also, categories or classes). Given a classification of a specific data set, there are four basic combinations of the actual data class and the assigned class: true positives (TP; actual positive and predicted positive), false positives (FP; actual negative and

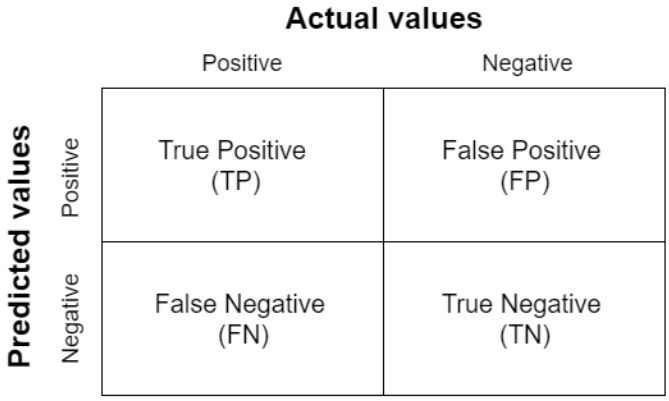


Figure 5.1: Combinations of actual data classes and assigned classes in binary classification

predicted positive), true negatives (TN; actual negative and predicted negative) and false negatives (FN; actual positive and predicted negative), where "positive" and "negative" represent two classes. These combinations can be arranged into a 2x2 contingency table as shown in Figure 5.1.

A calculation of the classification accuracy and F1 score is based on a number of occurrences of each combination in the classification. Accuracy is the ratio of a number of correct predictions to the total number of input samples. Accuracy of a binary classification is defined as:

$$accuracy = \frac{TP + TN}{P + N} \tag{5.13}$$

where $P = TP + FP$ and $N = TN + FN$. F1 score of binary classification is the harmonic average of the precision and the recall. Precision, recall and F1 score for the binary classification are defined as follows:

$$precision = \frac{TP}{TP + FP} \tag{5.14}$$

$$recall = \frac{TP}{TP + FN} \tag{5.15}$$

$$F1 = 2 \cdot \frac{precision \cdot recall}{precision + recall} \tag{5.16}$$

The metrics are shown graphically in Figure 5.2.

Multiclass (or multinomial) classification is the task of classifying the elements of a given set into three or more groups (categories, classes). It can be denoted as a function $g(x)$ that on input x returns one of the classes $1, ..., K$. Similarly to the binary classification, we can distinguish the combinations of actual data class and the predicted one. In this

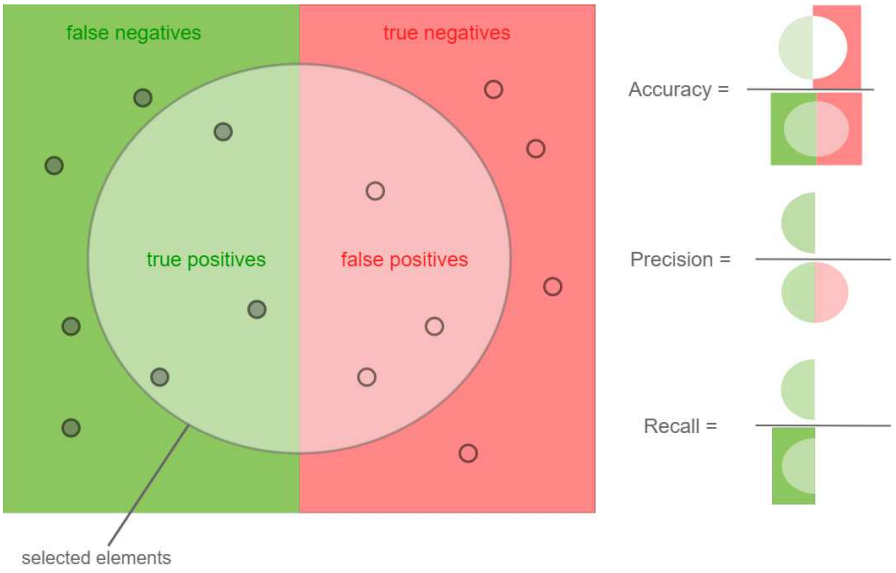


Figure 5.2: Classification performance metrics

case, we have true positives, false positives and false negatives associated with each class i. TP_i denotes the instances from the class i that are predicted to be in the class i, FP_i are instances belonging to another class, falsely predicted to be in the class i, and FN_i are the instances from the class i falsely predicted to be in some other class.

The accuracy of the multinomial classification is calculated as:

$$accuracy = \frac{1}{\eta} \cdot \sum_{i=1}^{K} \mathrm{TP}_i \tag{5.17}$$

where η is a total number of data instances. Note that Equation (5.17) is the general form that applies to the binary classification as well.

For multiclass classification, precision and recall can be micro- or macro-averaged, therefore also can be F1 score. The micro-averaged precision and recall are defined as follows:

$$precision_{micro} = \frac{\sum_{i=1}^{K} \mathrm{TP}_i}{\sum_{i=1}^{K} \mathrm{TP}_i + \mathrm{FP}_i} \tag{5.18}$$

$$recall_{micro} = \frac{\sum_{i=1}^{K} \mathrm{TP}_i}{\sum_{i=1}^{K} \mathrm{TP}_i + \mathrm{FN}_i} \tag{5.19}$$

Therefore, the micro F1 score is:

$$F1_{micro} = 2 \cdot \frac{precision_{micro} \cdot recall_{micro}}{precision_{micro} + recall_{micro}} \tag{5.20}$$

If micro F1 is a large value, this indicates that a classifier performs well overall, however it is not sensitive to the performance of individual classes. The macro-averaged precision and recall are:

$$precision_{macro} = \frac{1}{K}\sum_{i=1}^{K}\frac{\mathrm{TP}_i}{\mathrm{TP}_i + \mathrm{FP}_i} \tag{5.21}$$

$$recall_{macro} = \frac{1}{K}\sum_{i=1}^{K}\frac{\mathrm{TP}_i}{\mathrm{TP}_i + \mathrm{FN}_i} \tag{5.22}$$

Hence, the macro F1 score is:

$$F1_{macro} = 2 \cdot \frac{precision_{macro} \cdot recall_{macro}}{precision_{macro} + recall_{macro}} \tag{5.23}$$

A large macro F1 suggests that the classifier performs well for each individual class. The macro-average is therefore more suitable for data with an imbalanced class distribution. All of the above metrics reach their best value at 1 for the perfect classifier and worst at 0.

Experimental setup We used three datasets for the experiments - the Forest dataset, the Adult dataset and the Credit dataset. Therefore, three different classifications have been analysed. In the Forest dataset, the target attribute is *covertype* with 7 values, thus we are solving the multiclass classification problem. The accuracy is calculated as in Equation (5.17). Furthermore, we choose to use macro averages for calculating the F1 score (Equation (5.23)) to avoid the misleading nature of micro averages when the class distribution is imbalanced.

In the Adult dataset, we are predicting the binary attribute *income*. In the German Credit dataset, we are as well predicting the binary target attribute that classifies the data instance as good or bad credit risk. The binary classification is being performed and analysed for these two datasets and accordingly, the classification accuracy and F1 are calculated as Equation (5.13) and Equation (5.16).

For all of our experiments, we use 10-fold cross-validation to evaluate Machine Learning models. k-fold cross-validation is a resampling procedure used in Machine Learning to estimate the skill of a Machine Learning model on unseen data. It generally results in a less biased or less optimistic estimate of the model skill than other methods, such as a simple train/test split. The procedure starts by shuffling the dataset randomly and splitting it into k groups. In our experiments $k = 10$. Each unique group is taken as a hold out (test dataset) while the remaining $k - 1$ groups are taken as a training dataset. It fits a model on the training set and evaluates it on the test set, retaining the evaluation score for that group. At the end, we summarise the skill of the model using the sample of model evaluation scores. In the experiments, we record the average of these scores.

Generally, the goal was not to build models with best predictions but to compare the effectiveness of the fingerprinted data compared to the original. Therefore, there

is no advanced parameter optimisation included nor we dive into more complicated feature selection for building models, as it would shift the focus from the main goal. Still, we wanted to achieve the model performance levels rather close to the benchmark solutions available from numerous online sources[1,2] (the chosen datasets and the correlated classification problems are well known in the Machine Learning community). Therefore, it was important to find good hyperparameters for each of the classification tasks and each classifier. One possible solution besides the pure manual tuning is a random search over hyperparameters. We set up a grid of hyperparameter values and select a limited number of random combinations to train the model and score on validations data. This way we choose the combination of parameters giving the best score for our model.

We use Randomized Search with 10 iterations from the Python scikit-learn package[3] for tuning most important hyperparameters. Evaluation within the random search is done using 10-fold cross-validation and F1 score (macro F1 in case of a multiclass classification).

Finally, for our experiments, we use the following classifiers, all implemented in the Python scikit-learn package:

- Decision Tree
- k-NN (k Nearest Neighbours)
- Logistic regression
- Random Forest
- Gradient Boosting

The experimental process in the following sections has the following steps:

1. Use random search to tune the hyperparameters of the Decision Tree, k-NN, Logistic Regression, Random Forest and Gradient Boosting classifiers. In some of the experiments, we use only the subset of the classifiers above.
2. Train the model with the original dataset and score the classification accuracy and F1
3. For each fingerprinting parameter combination (γ, ξ) train the model with the fingerprinted dataset and score the classification accuracy and F1. Set the hyperparameters according to step 1.
4. Record the differences in the performance measures
5. Steps 3 and 4 are repeated 10 times to get the average values. In every experiment, we fingerprint the data with a random choice of buyer's id and secret key.

[1] https://www.kaggle.com/wenruliu/adult-income-dataset/kernels
[2] https://www.kaggle.com/c/forest-cover-type-prediction/kernels
[3] https://scikit-learn.org/stable/

5.2.1 Forest Cover Type

The first set of experiments is made using the biggest of the three datasets - Forest Cover Type. We use the AK Scheme for fingerprinting the numerical attributes of the dataset. The target attribute is *covertype* and all the others are used as an input for a prediction. We use the following classifiers and set the following hyperparameters:

- Decision Tree: $max\_depth = 5$, $criterion = entropy$
- Logistic Regression: $C = 100$
- Random Forest: $n\_estimators = 100$

The other hyperparameters are set to default values from the scikit-learn implementations of the classifiers.

The average differences are shown in Tables 5.6 to 5.8 for Decision Tree, Logistic Regression and Random Forest, respectively.

Table 5.6: Effects on F1 score and classification accuracy of a Decision Tree model trained with the Forest Cover Type dataset

	$\xi = 2$		$\xi = 4$		$\xi = 6$		average	
	F1	acc.	F1	acc.	F1	acc.	F1	acc.
$\gamma = 100$	0%	0%	0%	-0.01%	0%	-0.01%	0%	-0.01%
$\gamma = 50$	0%	-0.02%	0%	0%	0%	+0.01%	0%	0%
$\gamma = 25$	0%	-0.02%	+0.02%	+0.01%	-0.03%	-0.01%	0%	-0.01%
$\gamma = 12$	-0.01%	-0.02%	-0.01%	0%	-0.01%	-0.10%	-0.01%	-0.04%
$\gamma = 6$	-0.01%	0%	-0.04%	-0.01%	-0.19%	-0.11%	-0.08%	-0.04%
average	0%	-0.1%	-0.01%	0%	-0.05%	-0.04%	**-0.07%**	**-0.02%**

Table 5.7: Effects on F1 score and classification accuracy of a Logistic Regression model trained with the Forest Cover Type dataset

	$\xi = 2$		$\xi = 4$		$\xi = 6$		average	
	F1	acc.	F1	acc.	F1	acc.	F1	acc.
$\gamma = 100$	0%	0%	+0.01%	0%	-0.01%	+0.01%	0%	0%
$\gamma = 50$	-0.02%	0%	+0.01%	0%	-0.01%	+0.01%	-0.01%	0%
$\gamma = 25$	0%	0%	-0.01%	-0.01%	-0.05%	+0.02%	-0.02%	0%
$\gamma = 12$	0%	0%	-0.02%	0%	-0.11%	+0.02%	-0.04%	0%
$\gamma = 6$	0%	0%	-0.03%	0%	-0.14%	+0.03%	-0.06%	+0.01%
average	0%	0%	-0.01%	0%	-0.07%	+0.02%	**-0.03%**	**+0.01%**

The changes in performance are generally very small (in most of the cases the difference is in the 4th decimal place of the absolute values), therefore we represent the changes

Table 5.8: Effects on F1 score and classification accuracy of a Random Forest model trained with the Forest Cover Type dataset

	$\xi = 2$		$\xi = 4$		$\xi = 6$		average	
	F1	acc.	F1	acc.	F1	acc.	F1	acc.
$\gamma = 100$	+0.01%	-0.03%	+0.01%	-0.05%	-0.06%	-0.08%	-0.01%	-0.05%
$\gamma = 50$	0%	0%	+0.01%	+0.02%	-0.02%	-0.03%	0%	0%
$\gamma = 25$	0%	-0.01%	-0.07%	-0.03%	-0.05%	-0.03%	-0.04%	-0.02%
$\gamma = 12$	-0.02%	0%	-0.01%	0%	-0.03%	-0.05%	-0.02%	-0.02%
$\gamma = 6$	-0.04%	-0.08%	-0.01%	-0.03%	0%	-0.01%	-0.02%	-0.04%
average	-0.01%	-0.03%	-0.01%	-0.02%	-0.03%	-0.04%	**-0.02%**	**-0.03%**

as percentages in range (0-100%). All of the results roughly follow the rule that the performance measures decrease when γ decreases, i.e. more marks introduced in data. We can observe this behaviour in the last columns of every table where we calculate the average F1 score and accuracy for a fixed γ Furthermore, a general rule holds that the performance slightly drops for larger ξ values, i.e. more bits available for fingerprinting. This is due to larger distortions of particular values in the data. This is shown in the last row of every table where we average out the F1 score and accuracy for a fixed ξ. Every classifier from the experiments behaves very similarly in the means of performance decrease. Overall average F1 score and accuracy for every classifier is calculated from the experiment results and presented the bottom rightmost cell of each table.

On the Forest Cover Type data, we can conclude that the differences observed when using the Decision Tree classifier (see Table 5.6) are rather minute, and would not constitute a noticeable degradation of effectiveness. The trend is the same also for other classifiers, as can be seen in Table 5.7 for Logistic Regression, and Table 5.8 for Random Forest. In a few cases, the classification results obtained even improved, for example, the accuracy of Logistic Regression trained by data fingerprinted using $\xi = 6$, though by the same rather marginal order of magnitude as the observed decline.

5.2.2 Adult (Census Data)

We make the set of experiments using the Adult dataset. The dataset is mostly composed of categorical values, thus we use the fingerprinting scheme for categorical data from Section 3.7. The target of the classification task is *income*. All other attributes (including numerical, although we do not fingerprint them) are used for as an input for a classifier. We use Decision Tree, k-NN, Logistic Regression and Gradient Boosting. The hyperparameters are set as follows:

- k-NN: $n\_neighbors = 19$
- Logistic Regression: $solver = liblinear$, $C = 20$
- Random Forest: $n\_estimators = 200$, $max\_depth = 15$, $criterion = gini$

- Gradient Boosting: $n\_estimators := 40$, $max\_depth = 8$, $loss = deviance$, $criterion = mse$

The differences in F1 and accuracy scores (on a scale $[0, 100]\%$) between original and fingerprinted Adult dataset for Decision Tree, k-NN, Logistic Regression, Random Forest and Gradient Boosting are shown in Tables 5.9 to 5.12.

Table 5.9: Effects on F1 score and classification accuracy of a k-NN model trained with the Adult dataset

	$\xi = 1$		$\xi = 2$		$\xi = 4$		$\xi = 6$		average	
	F1	acc.	F1	acc.	F1	acc.	F1	acc.	F1	acc.
$\gamma = 50$	+0.05%	+0.03%	-0.10%	-0.05%	-0.06%	-0.02%	-0.02%	+0.01%	-0.03%	-0.01%
$\gamma = 25$	-0.10%	-0.05%	+0.05%	+0.02%	+0.07%	+0.03%	-0.02%	+0.03%	0%	-0.01%
$\gamma = 12$	-0.32%	-0.19%	-0.10%	-0.06%	+0.02%	+0.03%	-0.20%	-0.04%	-0.15%	-0.07%
$\gamma = 6$	-0.70%	-0.42%	-0.50%	-0.22%	-0.36%	-0.15%	-0.60%	-0.21%	-0.54%	-0.25%
$\gamma = 3$	-1.79%	-1.02%	-0.70%	-0.36%	-0.61%	-0.22%	-0.81%	-0.32%	-0.98%	-0.48%
average	-0.57%	-0.33%	-0.27%	-0.13%	-0.19%	-0.07%	-0.33%	-0.11%	**-0.34%**	**-0.16%**

Table 5.10: Effects on F1 score and classification accuracy of a Logistic Regression model trained with the Adult dataset

	$\xi = 1$		$\xi = 2$		$\xi = 4$		$\xi = 6$		average	
	F1	acc.	F1	acc.	F1	acc.	F1	acc.	F1	acc.
$\gamma = 50$	-0.15%	-0.07%	-0.02%	-0.01%	-0.07%	-0.03%	-0.03%	-0.02%	-0.07%	-0.03%
$\gamma = 25$	-0.25%	-0.14%	-0.13%	-0.06%	-0.10%	-0.06%	-0.14%	-0.06%	-0.16%	-0.08%
$\gamma = 12$	-0.46%	-0.22%	-0.27%	-0.12%	-0.12%	-0.08%	-0.39%	-0.15%	-0.31%	-0.14%
$\gamma = 6$	-0.68%	-0.38%	-0.41%	-0.22%	-0.46%	-0.19%	-0.80%	-0.33%	-0.59%	-0.28%
$\gamma = 3$	-2.12%	-1.01%	-1.08%	-0.52%	-0.75%	-0.32%	-1.33%	-0.62%	-1.32%	-0.62%
average	-0.73%	-0.36%	-0.38%	-0.19%	-0.25%	-0.14%	-0.54%	-0.24%	**-0.49%**	**-0.23%**

Table 5.11: Effects on F1 score and classification accuracy of a Random Forest model trained with the Adult dataset

	$\xi = 1$		$\xi = 2$		$\xi = 4$		$\xi = 6$		average	
	F1	acc.	F1	acc.	F1	acc.	F1	acc.	F1	acc.
$\gamma = 50$	-0.04%	+0.06%	-0.40%	-0.10%	-0.02%	+0.02%	-0.37%	-0.09%	-0.21%	-0.03%
$\gamma = 25$	-0.28%	-0.13%	-0.20%	-0.03%	-0.29%	-0.08%	-0.56%	-0.15%	-0.33%	-0.10%
$\gamma = 12$	-0.59%	-0.23%	-0.63%	-0.23%	-0.13%	+0.02%	-0.34%	-0.09%	-0.42%	-0.13%
$\gamma = 6$	-0.68%	-0.31%	-0.66%	-0.20%	-0.33%	-0.08%	-0.76%	-0.24%	-0.61%	-0.27%
$\gamma = 3$	-2.26%	-1.02%	-1.04%	-0.37%	-1.24%	-0.40%	-1.01%	-0.35%	-1.39%	-0.54%
average	-0.77%	-0.33%	-0.59%	-0.19%	-0.40%	-0.10%	-0.61%	-0.18%	**-0.69%**	**-0.20%**

Overall, each classifier shows a decrease in performance when the fingerprint is applied. Except for a few cases where the performance slightly improves, both F1 and accuracy decrease up to approximately 2%. The average difference for each classifier can be seen in the respective table in the rightmost cell of the last row in bold characters. The largest average difference for the entire classifier is the change in F1 score of Gradient Boosting,

Table 5.12: Effects on F1 score and classification accuracy of a Gradient Boosting model trained with the Adult dataset

	$\xi = 1$		$\xi = 2$		$\xi = 4$		$\xi = 6$		average	
	F1	acc.	F1	acc.	F1	acc.	F1	acc.	F1	acc.
$\gamma = 50$	-0.12%	-0.06%	-0.32%	-0.16%	+0.02%	-0.01%	-0.17%	-0.07%	-0.15%	-0.08%
$\gamma = 25$	-0.40%	-0.19%	-0.43%	-0.20%	-0.13%	-0.08%	-0.30%	-0.10%	-0.32%	-0.14%
$\gamma = 12$	-0.71%	-0.34%	-0.51%	-0.18%	-0.42%	-0.18%	-0.29%	-0.11%	-0.48%	-0.20%
$\gamma = 6$	-0.85%	-0.45%	-0.74%	-0.35%	-0.39%	-0.20%	-0.69%	-0.30%	-0.67%	-0.33%
$\gamma = 3$	-2.60%	-1.22%	-1.18%	-0.59%	-0.76%	-0.36%	-0.59%	-0.27%	-1.28%	-0.61%
average	-0.94%	-0.45%	-0.64%	-0.30%	-0.34%	-0.17%	-0.41%	-0.17%	**-0.58%**	**-0.27%**

-0.58% (Table 5.12). Therefore, we can argue that changes in F1 and accuracy in this range are rather acceptable. This is, of course, dependable on the use case. However, we need to consider that fingerprinting necessarily changes the values in the dataset and the performance can not be exactly the same as when the original dataset is used. The occurrences of the positive differences (see, for example, Table 5.10, $\gamma = 50$, $\xi = 1$) are random and minute, and therefore concluding that a fingerprint in the data improves the performance of the Machine Learning model is a rule for certain cases would be wrong.

In the last row and the last column of every table, we calculate the average difference for the fixed ξ or γ, respectively, to see the effects of these parameters more easily. It is hard to detect any pattern between average values of F1/accuracy and ξ value for any of the classifiers. This behaviour can be expected because bigger ξ do not imply "more change" in values for this fingerprinting scheme compared to the schemes for numerical values. For instance, using $\xi = 3$ in fingerprinting could change a value "9" to "13" (difference of 4), while using $\xi = 1$ could only (and at most!) change it to "8" (difference of 1). This behaviour does not apply to categorical data since the change from any categorical value to another is unit. Therefore, the effect of the parameter ξ on the performance of a model trained using fingerprinted data in classification is random.

On the other hand, we can see the trend of a decrease in performance for smaller values of γ in the last column of the tables. Both average F1 and accuracy gradually decrease for smaller γ, i.e. more marks in the dataset. It is the case for a majority of the particular cases with one fixed value of ξ. For example, the entire experimental results with Logistic Regression, in Table 5.10, have a perfect inversely proportional relation between a value of γ and absolute difference in F1 score or accuracy.

On the Adult dataset we can conclude that, from the classification point of view, the utility of data is preserved after applying the proposed fingerprinting technique for categorical data. The performance drops are not significant and can be controlled by the parameter γ.

5.2.3 German Credit Data

With the last set of experiments, we analyse the utility of the fingerprinted data that contains a mixture of numerical and non-numerical attributes. We use the German Credit dataset and apply both AK Scheme to the numerical values and the naive fingerprinting technique for categorical data to non-numerical. We unify these two processes into one fingerprinting process since these techniques share the algorithmic steps, except for the modification for marking the categorical values. We use Decision Tree, k-NN, Logistic Regression and Random Forest for the classification and follow the usual procedure. We find with the random search that the best hyperparameters for the model trained with original data are as follows:

- Decision Tree: $max\_depth = 2$, $criterion = entropy$
- k-NN: $n\_neighbors = 14$
- Logistic Regression: $solver = newton - cg$, $C = 70$
- Random Forest: $n\_estimators = 85$, $max\_depth = 10$, $criterion = gini$

Tables 5.13 to 5.16 show the resulting differences in F1 and accuracy scores (on a scale $[0, 100]\%$) between original and fingerprinted German Credit data for Decision Tree, k-NN, Logistic Regression and Random Forest, respectively.

Table 5.13: Effects on F1 score and classification accuracy of a Decision Tree model trained with the German Credit dataset

	$\xi = 1$		$\xi = 2$		$\xi = 4$		$\xi = 6$		average	
	F1	acc.	F1	acc.	F1	acc.	F1	acc.	F1	acc.
$\gamma = 12$	0%	0%	-0.07%	-0.10%	-0.07%	-0.10%	0%	0%	-0.04%	-0.05%
$\gamma = 9$	0%	0%	0%	0%	-0.14%	-0.20%	-0.14%	-0.20%	-0.07%	-0.10%
$\gamma = 6$	0%	0%	-0.07%	-0.10%	-0.14%	-0.20%	-0.14%	-0.20%	-0.09%	-0.13%
$\gamma = 3$	0%	0%	-0.14%	-0.20%	-0.14%	-0.20%	-0.42%	-0.60%	-0.18%	-0.25%
average	0%	0%	-0.07%	-0.10%	-0.12%	-0.18%	-0.18%	-0.25%	**-0.09%**	**-0.13%**

Table 5.14: Effects on F1 score and classification accuracy of a k-NN model trained with the German Credit dataset

	$\xi = 1$		$\xi = 2$		$\xi = 4$		$\xi = 6$		average	
	F1	acc.	F1	acc.	F1	acc.	F1	acc.	F1	acc.
$\gamma = 12$	0.0%	0.0%	-0.14%	-0.20%	-0.07%	-0.10%	0.0%	0.0%	-0.05%	-0.08%
$\gamma = 9$	0.0%	0.0%	-0.05%	-0.10%	-0.14%	-0.20%	-0.22%	-0.30%	-0.10%	-0.15%
$\gamma = 6$	0.0%	0.0%	-0.27%	-0.40%	-0.14%	-0.20%	-0.22%	-0.30%	-0.16%	-0.23%
$\gamma = 3$	0.0%	0.0%	-0.09%	-0.10%	-0.39%	-0.60%	-0.45%	-0.60%	-0.23%	-0.33%
average	0%	0%	-0.14%	-0.20%	-0.19%	-0.28%	-0.22%	-0.30%	**-0.14%**	**-0.19%**

Table 5.15: Effects on F1 score and classification accuracy of a Logistic Regression model trained with the German Credit dataset

	$\xi = 1$		$\xi = 2$		$\xi = 4$		$\xi = 6$		average	
	F1	acc.	F1	acc.	F1	acc.	F1	acc.	F1	acc.
$\gamma = 12$	-0.02%	0.0%	-0.45%	-0.61%	-0.02%	+0.09%	+0.17%	+0.30%	-0.08%	-0.06%
$\gamma = 9$	-0.04%	0.0%	-0.27%	-0.30%	-0.38%	-0.60%	-0.25%	-0.50%	-0.24%	-0.35%
$\gamma = 6$	-0.05%	-0.10%	-0.42%	-0.71%	-0.55%	-0.80%	-0.25%	-0.40%	-0.32%	-0.50%
$\gamma = 3$	-0.17%	-0.30%	-0.26%	-0.41%	-0.21%	-0.50%	-0.29%	-0.41%	-0.23%	-0.41%
average	-0.07%	-0.10%	-0.35%	-0.51%	-0.29%	-0.45%	-0.16%	-0.25%	**-0.22%**	**-0.33%**

Table 5.16: Effects on F1 score and classification accuracy of a decision Tree model trained with the Random Forest dataset

	$\xi = 1$		$\xi = 2$		$\xi = 4$		$\xi = 6$		average	
	F1	acc.	F1	acc.	F1	acc.	F1	acc.	F1	acc.
$\gamma = 12$	-0.47%	-0.70%	-0.81%	-1.20%	-0.07%	-0.31%	-0.21%	-0.50%	-0.39%	-0.68%
$\gamma = 9$	-0.50%	-0.80%	-0.34%	-0.70%	-0.31%	-0.50%	-0.35%	-0.61%	-0.38%	-0.62%
$\gamma = 6$	-0.82%	-1.30%	-0.42%	-0.90%	-0.44%	-0.70%	-0.30%	-0.41%	-0.50%	-0.83%
$\gamma = 3$	-0.79%	-1.40%	-0.78%	-1.30%	-1.23%	-2.10%	-0.85%	-1.20%	-0.91%	-1.50%
average	-0.65%	-1.05%	-0.59%	-1.03%	-0.51%	-0.90%	-0.43%	-0.68%	**-0.54%**	**-0.91%**

Similar to the Forest Cover Type dataset and Adult dataset, we can note that there are very small effects on the classification accuracy and F1 score using German Credit dataset.

In experiments, the classification accuracy and F1 score generally slightly decrease for smaller γ, i.e. by introducing more error, which is expected. The effect of parameter ξ is not obvious in all of the cases. We now have a mixture of numerical and non-numerical values. We have discussed in the previous sections that ξ affects the performance when numerical data is used, while with non-numerical it is not the case. For example, results from Decision Tree in Table 5.13 show the decrease in F1 and accuracy for larger ξ, while the results from Logistic Regression in Table 5.15 do not show such relation.

Generally, bigger errors introduced by fingerprinting using the naive technique for categorical data did not significantly affect the performance of any of the classifiers.

We further run experiments to analyse the second proposed technique for fingerprinting the categorical type of data. German Credit Data is fingerprinted using the approach of finding a fixed number of neighbours, with $k = 10$. We use the same set values for parameters ξ and γ as in the previous experiments. We trained the Logistic Regression and the Random Forest models. The differences of the performance measures, classification accuracy and F1 score, are shown in Tables 5.17 and 5.18.

The results are, generally, very similar to the results of the naive fingerprinting technique. The decrease in performance is in the range of approximately -1% for both classification accuracy and F1 score. The Logistic Regression gives overall slightly worse results in this

Table 5.17: Effects on F1 score and classification accuracy of a Logistic Regression model trained with the German Credit dataset fingerprinted with the neighbourhood-based technique

	$\xi = 1$		$\xi = 2$		$\xi = 4$		$\xi = 6$		average	
	F1	acc.	F1	acc.	F1	acc.	F1	acc.	F1	acc.
$\gamma = 12$	-0.19%	-0.20%	+0.03%	+0.09%	0%	-0.01%	-0.19%	-0.30%	-0.09%	-0.11%
$\gamma = 9$	-0.24%	-0.30%	-0.23%	-0.30%	0%	0%	-0.34%	-0.50%	-0.20%	-0.28%
$\gamma = 6$	-0.47%	-0.60%	-0.16%	-0.31%	-0.28%	-0.41%	-0.93%	-1.31%	-0.46%	-0.66%
$\gamma = 3$	-0.65%	-0.90%	-0.33%	-0.60%	+0.13%	+0.09%	-0.07%	-0.20%	-0.23%	-0.40%
average	-0.39%	-0.50%	-0.17%	-0.28%	-0.04%	-0.08%	-0.38%	-0.58%	**-0.25%**	**-0.36%**

Table 5.18: Effects on F1 score and classification accuracy of a Random Forest model trained with the German Credit dataset fingerprinted with the neighbourhood-based technique

	$\xi = 1$		$\xi = 2$		$\xi = 4$		$\xi = 6$		average	
	F1	acc.	F1	acc.	F1	acc.	F1	acc.	F1	acc.
$\gamma = 12$	-0.27%	-0.60%	-0.16%	-0.50%	-0.57%	-1.10%	-0.43%	-1.00%	-0.36%	-0.80%
$\gamma = 9$	-0.33%	-0.60%	-0.36%	-0.70%	-0.45%	-0.90%	-0.48%	-0.90%	-0.41%	-0.78%
$\gamma = 6$	-0.56%	-1.00%	-0.22%	-0.40%	-0.73%	-1.31%	-0.41%	-0.90%	-0.48%	-0.90%
$\gamma = 3$	-0.65%	-1.10%	-0.95%	-1.90%	-0.40%	-0.90%	+0.01	-0.20%	-0.50%	-1.03%
average	-0.45%	-0.82%	-0.42%	-0.88%	-0.54%	-1.05%	-0.33%	-0.75%	**-0.44%**	**-0.88%**

case, while for the Random Forest the performance is slightly better compared to the naive technique. The performance follows the same trend of a decrease when more marks are embedded in the data. See, for example, the last column in Table 5.18 where we recorded the average F1 and accuracy decrease for different values of ξ. The difference is larger for smaller values of γ, i.e. more marks in the data.

5.3 Summary

In this chapter, we deliver the analysis of the utility of the fingerprinted data. We analyse the utility from the two aspects. First, we measure the mean and variance of each attribute of the datasets before and after fingerprinting and compare them. Fingerprinting techniques for numerical data introduce almost no change in the mean and very little change in the variance. For the fingerprinting technique for non-numerical data where mean and variance are not applicable, we measure the quality by counting the number of changes.

Secondly, we analyse the effect of the fingerprint on the classification performance. We build Machine Learning models using the original dataset and record the performance scores. Then we record the performance scores of the models using the fingerprinted datasets, under the same set of hyperparameters. The performance measures we use are the classification accuracy and F1 score. Our experiments show that the decrease in performance is minute. The fingerprinting parameters, for example, the number of marks

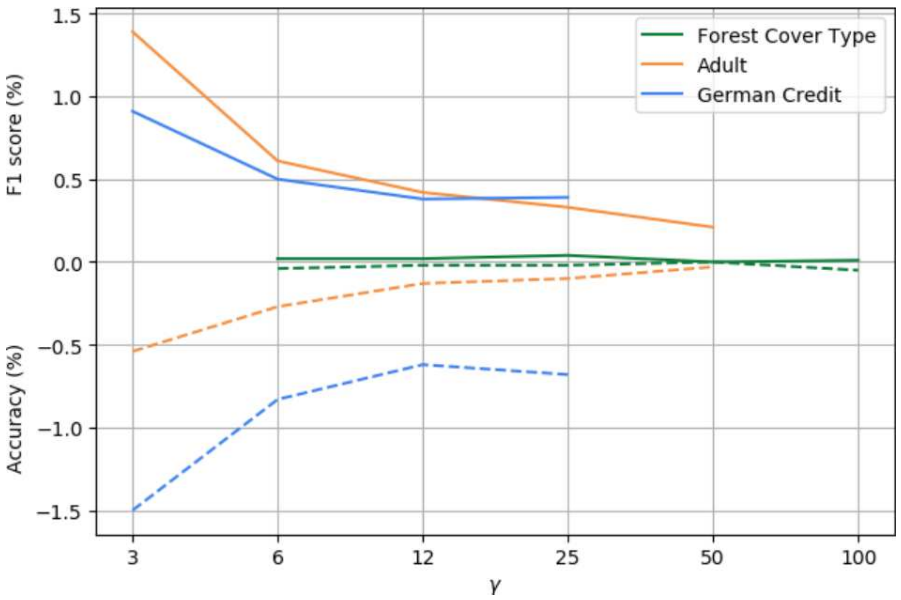


Figure 5.3: Error in performance measures of the Random Forest classifier

controlled by the parameter γ, influence the performance, however, the difference stays in the same range. The example of the performance decrease depending on γ is shown in Figure 5.3 for Random Forest and three different datasets. The F1 score differences are reflected over the x-axis for clarity. The trend of decrease in the performance error can be seen for fewer marks in data (bigger γ). Furthermore, we can see that the range of the errors, both accuracy and F1, are within the range [0,1.5]%, based on which we can argue that the error introduced by the fingerprint is negligible.

We obtain similar results with every other classifier, including Decision Tree, Logistic Regression, k-NN and Gradient Boosting. Our experiments with data fingerprinted with the second fingerprinting technique for categorical data based on finding the closest neighbourhood show very similar results to the naive approach. In future work, this technique will be more closely analysed, both from the robustness and the data utility point of view. In this case, the technique did not show distinctively better results than the naive approach. However, this might be case-specific and differ for the other choices of the parameter k that defines the size of the neighbourhood.

CHAPTER 6

Conclusion

This thesis elaborates the concept of fingerprinting relational databases. We implement and analyse three fingerprinting techniques for relational data containing numerical values - AK Scheme, Block Scheme and Two-level Fingerprinting Scheme. The real-world datasets oftentimes contain non-numerical attributes. Therefore, we propose, implement and analyse two approaches for fingerprinting non-numerical data. The naive approach, inspired by techniques for fingerprinting numerical data, first converts the categorical attributes to numerical and then applies random marks on the chosen values. The second approach goes towards minimising contextual inconsistencies in the data after fingerprinting. Even though this technique is not blind, it might be a better solution for fingerprinting data with correlated categorical attributes than marking the values randomly.

We analyse the robustness of the techniques against the attacks. We analyse it theoretically and empirically, using the implemented techniques and a broad set of parameters. Attacks that modify the dataset without any background knowledge of where the marks are embedded and without the access to other copies of fingerprinted data (e.g. subset attack, superset attack, bit-flipping attack), are shown to be avoidable by careful parameter tuning. In most of the cases, the robustness is increased by increasing the number of marks in the data. If a smaller dataset is being fingerprinted, a fingerprinting scheme that creates shorter fingerprints is more robust.

The AK Scheme appears to be the most robust technique among the analysed ones. This is beneficial for our choice that the proposed scheme for categorical data follows the same steps as the AK Scheme. The Two-level fingerprinting technique is very useful because of the verification of the ownership and the source of the leakage in two separate processes. The ownership verification level is much more robust to any kind of attack, which allows the owner to prove ownership even in cases when the exact fingerprint of the receiver cannot be extracted. The Block Scheme is inspired by a useful fingerprinting technique for fingerprinting images, however, does not appear to be very robust in its adaptation to

fingerprinting technique for relational datasets. The proposed technique for categorical data is less robust against the attacks than AK Scheme or Two-level Scheme, however still applicable.

We further analyse how the modifications in data affect its utility. The modifications in the mean and variance of the attributes are rather minute. More marks in data and more allowed bits for embedding the mark result in bigger variance errors. The utility is further measured in the assumed scenario where the fingerprinted data is used for training a Machine Learning model that is making some prediction. We use a variety of Machine Learning classifiers, including Decision Tree, Logistic Regression, k-NN, Random Forest and Gradient Boosting, and experiment with three different datasets. The resulting decrease of performance when compared to the model trained with original data is around 1%, mostly less (assuming the accuracy and F1 score are measured in the range [0,100]%). Therefore, the predictions are rather accurate when the fingerprinted data is used.

The parameters that would, for example, increase the robustness of the scheme, decrease the utility. We show in Table 6.1 the effects of the main fingerprinting parameters on different types of attacks and the utility.

Table 6.1: Impact of parameters on robustness against attacks resp. on data utility

↑	ω	ξ	τ	L	N
Misdiagnosis false hit	↑		↑	↑	↓
Subset Attack	↑		↓	↓	
Bit-flipping Attack	↑	↑	↓	↓	
Additive Attack	↑	↓	↓		
Utility	↓	↓			

For example, increasing the number of marks in the data ω would increase the scheme's robustness against misdiagnosis false hit, subset attack, bit-flipping attack and additive attack, while decreasing the utility. Increasing the number of LSBs available for fingerprinting ξ, does not affect the robustness against misdiagnosis false hit and subset attack, increases robustness against bit-flipping attack, but decreases the robustness against the additive attack and the utility.

We show that the fingerprint does not affect the utility of the data too much. However, it depends on the use case if these, although minor, effects are acceptable.

List of Figures

List of Tables

List of Algorithms

Bibliography

[1] Y. Li, V. Swarup, and S. Jajodia, "Fingerprinting relational databases: Schemes and specialties," *IEEE Transactions on Dependable and Secure Computing*, vol. 2, no. 1, pp. 34–45, 2005.

[2] S. Liu, S. Wang, R. H. Deng, and W. Shao, "A block oriented fingerprinting scheme in relational database," in *International Conference on Information Security and Cryptology*, pp. 455–466, Springer, 2004.

[3] F. Guo, J. Wang, and D. Li, "Fingerprinting relational databases," in *Proceedings of the 2006 ACM symposium on Applied computing*, pp. 487–492, ACM, 2006.

[4] C. Constantin, D. Gross-Amblard, and M. Guerrouani, "Watermill: an optimized fingerprinting system for highly constrained data," in *Proceedings of the 7th workshop on Multimedia and security*, pp. 143–155, ACM, 2005.

[5] R. Halder, S. Pal, and A. Cortesi, "Watermarking techniques for relational databases: Survey, classification and comparison.," *J. UCS*, vol. 16, no. 21, pp. 3164–3190, 2010.

[6] I. J. Cox, J. Kilian, F. T. Leighton, and T. Shamoon, "Secure spread spectrum watermarking for multimedia," *IEEE transactions on image processing*, vol. 6, no. 12, pp. 1673–1687, 1997.

[7] F. A. Petitcolas and S. Katzenbeisser, *Information Hiding Techniques for Steganography and Digital Watermarking (Artech House Computer Security Series)*. Artech House, 2000.

[8] S.-J. Lee and S.-H. Jung, "A survey of watermarking techniques applied to multimedia," in *ISIE 2001. 2001 IEEE International Symposium on Industrial Electronics Proceedings (Cat. No. 01TH8570)*, vol. 1, pp. 272–277, IEEE, 2001.

[9] M. Wu, W. Trappe, Z. J. Wang, and K. R. Liu, "Collusion-resistant fingerprinting for multimedia," *IEEE Signal Processing Magazine*, vol. 21, no. 2, pp. 15–27, 2004.

[10] J. O'Ruanaidh, W. Dowling, and F. Boland, "Watermarking digital images for copyright protection," *IEE Proceedings-Vision, Image and Signal Processing*, vol. 143, no. 4, pp. 250–256, 1996.

[11] F. Hartung and B. Girod, "Watermarking of uncompressed and compressed video," *Signal processing*, vol. 66, no. 3, pp. 283–301, 1998.

[12] L. Boney, A. H. Tewfik, and K. N. Hamdy, "Digital watermarks for audio signals," in *Proceedings of the Third IEEE International Conference on Multimedia Computing and Systems*, pp. 473–480, IEEE, 1996.

[13] M. D. Swanson, B. Zhu, A. H. Tewfik, and L. Boney, "Robust audio watermarking using perceptual masking," *Signal processing*, vol. 66, no. 3, pp. 337–355, 1998.

[14] V. M. Potdar, S. Han, and E. Chang, "A survey of digital image watermarking techniques," in *INDIN'05. 2005 3rd IEEE International Conference on Industrial Informatics, 2005.*, pp. 709–716, IEEE, 2005.

[15] N. Nikolaidis and I. Pitas, "Robust image watermarking in the spatial domain," *Signal processing*, vol. 66, no. 3, pp. 385–403, 1998.

[16] M. Piec and A. Rauber, "Real-time screen watermarking using overlaying layer," in *2014 Ninth International Conference on Availability, Reliability and Security*, pp. 561–570, IEEE, 2014.

[17] M. A. Suhail and M. S. Obaidat, "Digital watermarking-based dct and jpeg model," *IEEE transactions on instrumentation and measurement*, vol. 52, no. 5, pp. 1640–1647, 2003.

[18] V. Solachidis and L. Pitas, "Circularly symmetric watermark embedding in 2-d dft domain," *IEEE transactions on image processing*, vol. 10, no. 11, pp. 1741–1753, 2001.

[19] R. B. Wolfgang, C. I. Podilchuk, and E. J. Delp, "Perceptual watermarks for digital images and video," *Proceedings of the IEEE*, vol. 87, no. 7, pp. 1108–1126, 1999.

[20] M. D. Swanson, B. Zhu, and A. H. Tewfik, "Multiresolution scene-based video watermarking using perceptual models," *IEEE Journal on selected areas in communications*, vol. 16, no. 4, pp. 540–550, 1998.

[21] I.-K. Kang, D.-H. Im, H.-K. Lee, and Y.-H. Suh, "Implementation of real-time watermarking scheme for high-quality video," in *Proceedings of the 8th workshop on Multimedia and security*, pp. 124–129, ACM, 2006.

[22] N. F. Maxemchuk, "Electronic document distribution," *AT&T technical journal*, vol. 73, no. 5, pp. 73–80, 1994.

[23] M. J. Atallah, V. Raskin, M. Crogan, C. Hempelmann, F. Kerschbaum, D. Mohamed, and S. Naik, "Natural language watermarking: Design, analysis, and a proof-of-concept implementation," *International Workshop on Information Hiding*, pp. 185–200, 2001.

[24] M. J. Atallah, C. J. McDonough, V. Raskin, and S. Nirenburg, "Natural language processing for information assurance and security: an overview and implementations," *NSPW*, pp. 51–65, 2000.

[25] M. J. Atallah and S. S. Wagstaff, "Watermarking with quadratic residues," *Security and Watermarking of Multimedia Contents*, vol. 3657, pp. 283–289, 1999.

[26] M. J. Atallah, V. Raskin, C. F. Hempelmann, M. Karahan, R. Sion, U. Topkara, and K. E. Triezenberg, "Natural language watermarking and tamperproofing," in *International workshop on information hiding*, pp. 196–212, Springer, 2002.

[27] J. Palsberg, S. Krishnaswamy, M. Kwon, D. Ma, Q. Shao, and Y. Zhang, "Experience with software watermarking," in *Proceedings 16th Annual Computer Security Applications Conference (ACSAC'00)*, pp. 308–316, IEEE, 2000.

[28] C. Collberg and C. Thomborson, "Software watermarking: Models and dynamic embeddings," in *Proceedings of the 26th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, pp. 311–324, ACM, 1999.

[29] M. K. Rathva and G. Sahani, "Watermarking relational databases," *International Journal of Computer Science, Engineering and Applications*, vol. 3, no. 1, p. 71, 2013.

[30] B. B. Mehta and H. D. Aswar, "Watermarking for security in database: A review," in *2014 Conference on IT in Business, Industry and Government (CSIBIG)*, pp. 1–6, IEEE, 2014.

[31] S. Iftikhar, M. Kamran, and Z. Anwar, "A survey on reversible watermarking techniques for relational databases," *Security and Communication Networks*, vol. 8, no. 15, pp. 2580–2603, 2015.

[32] M. Kamran and M. Farooq, "A comprehensive survey of watermarking relational databases research," *arXiv preprint arXiv:1801.08271*, 2018.

[33] R. Agrawal and J. Kiernan, "Watermarking relational databases," in *VLDB'02: Proceedings of the 28th International Conference on Very Large Databases*, pp. 155–166, Elsevier, 2002.

[34] R. Agrawal, P. J. Haas, and J. Kiernan, "A system for watermarking relational databases," in *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, pp. 674–674, ACM, 2003.

[35] R. Agrawal, P. J. Haas, and J. Kiernan, "Watermarking relational data: framework, algorithms and analysis," *The VLDB Journal—The International Journal on Very Large Data Bases*, vol. 12, no. 2, pp. 157–169, 2003.

[36] Y. Li, V. Swarup, and S. Jajodia, "Constructing a virtual primary key for fingerprinting relational data," in *Proceedings of the 3rd ACM workshop on Digital rights management*, pp. 133–141, ACM, 2003.

[37] J. Lafaye, "An analysis of database watermarking security," in *Third International Symposium on Information Assurance and Security*, pp. 462–467, IEEE, 2007.

[38] Z. Qin, Y. Ying, L. Jia-jin, and L. Yi-shu, "Watermark based copyright protection of outsourced database," in *2006 10th International Database Engineering and Applications Symposium (IDEAS'06)*, pp. 301–308, IEEE, 2006.

[39] G. Gupta and J. Pieprzyk, "Database relation watermarking resilient against secondary watermarking attacks," in *International Conference on Information Systems Security*, pp. 222–236, Springer, 2009.

[40] X. Xiao, X. Sun, and M. Chen, "Second-lsb-dependent robust watermarking for relational database," in *Third International Symposium on Information Assurance and Security*, pp. 292–300, IEEE, 2007.

[41] C. Wang, J. Wang, M. Zhou, G. Chen, and D. Li, "Atbam: An arnold transform based method on watermarking relational data," in *2008 International Conference on Multimedia and Ubiquitous Engineering (MUE 2008)*, pp. 263–270, IEEE, 2008.

[42] Z. Hu, Z. Cao, and J. Sun, "An image based algorithm for watermarking relational databases," in *2009 International Conference on Measuring Technology and Mechatronics Automation*, vol. 1, pp. 425–428, IEEE, 2009.

[43] X. Zhou, M. Huang, and Z. Peng, "An additive-attack-proof watermarking mechanism for databases' copyrights protection using image," in *Proceedings of the 2007 ACM symposium on Applied computing*, pp. 254–258, ACM, 2007.

[44] H. Wang, X. Cui, and Z. Cao, "A speech based algorithm for watermarking relational databases," in *2008 International Symposiums on Information Processing*, pp. 603–606, IEEE, 2008.

[45] X. Cui and H. Cui, "The approach for optimization in watermark signal of relational databases by using genetic algorithms," in *2008 International Conference on Computer Science and Information Technology*, pp. 448–452, IEEE, 2008.

[46] Y. Zhang, X. Niu, and D. Zhao, "A method of protecting relational databases copyright with cloud watermark," *International Journal of Information and Communication Engineering*, vol. 1, no. 7, pp. 337–341, 2005.

[47] K. Huang, M. Yue, P. Chen, Y. He, and X. Chen, "A cluster-based watermarking technique for relational database," in *2009 First International Workshop on Database Technology and Applications*, pp. 107–110, IEEE, 2009.

[48] Y. Zhang, X. Niu, D. Zhao, J. Li, and S. Liu, "Relational databases watermark technique based on content characteristic," in *First International Conference on Innovative Computing, Information and Control-Volume I (ICICIC'06)*, vol. 3, pp. 677–680, IEEE, 2006.

[49] H. Guo, Y. Li, A. Liu, and S. Jajodia, "A fragile watermarking scheme for detecting malicious modifications of database relations," *Information Sciences*, vol. 176, no. 10, pp. 1350–1378, 2006.

[50] R. Sion, "Proving ownership over categorical data," in *Proceedings. 20th International Conference on Data Engineering*, pp. 584–595, IEEE, 2004.

[51] R. Sion, M. Atallah, and S. Prabhakar, "Rights protection for categorical data," *IEEE Transactions on Knowledge and Data Engineering*, vol. 17, no. 7, pp. 912–926, 2005.

[52] E. Bertino, B. C. Ooi, Y. Yang, and R. H. Deng, "Privacy and ownership preserving of outsourced medical data," in *21st International Conference on Data Engineering (ICDE'05)*, pp. 521–532, IEEE, 2005.

[53] A. Al-Haj and A. Odeh, "Robust and blind watermarking of relational database systems," 2008.

[54] V. Pournaghshband, "A new watermarking approach for relational data," in *Proceedings of the 46th annual southeast regional conference on XX*, pp. 127–131, ACM, 2008.

[55] V. Prasannakumari, "A robust tamperproof watermarking for data integrity in relational databases," *Research Journal of Information Technology*, vol. 1, no. 3, pp. 115–121, 2009.

[56] Y. Li, H. Guo, and S. Jajodia, "Tamper detection and localization for categorical data using fragile watermarks," in *Proceedings of the 4th ACM workshop on Digital rights management*, pp. 73–82, ACM, 2004.

[57] S. Bhattacharya and A. Cortesi, "Distortion-free authentication watermarking.," in *International Conference on Software and Data Technologies*, pp. 205–219, Springer, 2010.

[58] M.-H. Tsai, H.-Y. Tseng, and C.-Y. Lai, "A database watermarking technique for temper detection," in *9th Joint International Conference on Information Sciences (JCIS-06)*, Atlantis Press, 2006.

[59] Y. Li and R. H. Deng, "Publicly verifiable ownership protection for relational databases," in *Proceedings of the 2006 ACM Symposium on Information, computer and communications security*, pp. 78–89, ACM, 2006.

[60] S. Bhattacharya and A. Cortesi, "A generic distortion free watermarking technique for relational databases," in *International Conference on Information Systems Security*, pp. 252–264, Springer, 2009.

[61] I. Kamel, "A schema for protecting the integrity of databases," *computers & security*, vol. 28, no. 7, pp. 698–709, 2009.

[62] T. K. Das and S. Maitra, "A robust block oriented watermarking scheme in spatial domain," in *International Conference on Information and Communications Security*, pp. 184–196, Springer, 2002.

[63] J. Lafaye, D. Gross-Amblard, C. Constantin, and M. Guerrouani, "Watermill: An optimized fingerprinting system for databases under constraints," *IEEE Transactions on Knowledge and Data Engineering*, vol. 20, no. 4, pp. 532–546, 2008.

[64] F. Guo, J. Wang, Z. Zhang, X. Ye, and D. Li, "An improved algorithm to watermark numeric relational data," in *International Workshop on Information Security Applications*, pp. 138–149, Springer, 2005.

[65] M. Zhou, J. Wang, C. Wang, and D. Li, "A novel fingerprinting architecture for relational data," in *2007 Inaugural IEEE-IES Digital EcoSystems and Technologies Conference*, pp. 477–480, IEEE, 2007.

[66] P. Kieseberg, S. Schrittwieser, M. Mulazzani, I. Echizen, and E. Weippl, "An algorithm for collusion-resistant anonymization and fingerprinting of sensitive microdata," *Electronic Markets*, vol. 24, pp. 113–124, Jun 2014.

[67] L. Sweeney, "K-anonymity: A model for protecting privacy," *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, vol. 10, pp. 557–570, Oct. 2002.

[68] A. C. Yao, "Theory and application of trapdoor functions," in *23rd Annual Symposium on Foundations of Computer Science (sfcs 1982)*, pp. 80–91, IEEE, 1982.

[69] J. Kelsey, B. Schneier, and N. Ferguson, "Yarrow-160: Notes on the design and analysis of the yarrow cryptographic pseudorandom number generator," in *International Workshop on Selected Areas in Cryptography*, pp. 13–33, Springer, 1999.

[70] N. Ferguson, B. Schneier, and T. Kohno, "Generating randomness," *Cryptography Engineering: Design Principles and Practical Applications*, pp. 135–161, 2015.

[71] "Microsoft documentation for CryptGenRandom." `https://docs.microsoft.com/en-us/windows/win32/api/wincrypt/nf-wincrypt-cryptgenrandom`. Accessed: 2019-08-01.

[72] R. Rivest, "The md5 message-digest algorithm," 1992.

[73] D. Eastlake and P. Jones, "Us secure hash algorithm 1 (sha1)," 2001.

[74] D. Boneh and J. Shaw, "Collusion-secure fingerprinting for digital data," *IEEE Transactions on Information Theory*, vol. 44, no. 5, pp. 1897–1905, 1998.

[75] H.-J. Guth and B. Pfitzmann, "Error-and collusion-secure fingerprinting for digital data," in *International Workshop on Information Hiding*, pp. 134–145, Springer, 1999.

[76] B. Pfitzmann and A.-R. Sadeghi, "Coin-based anonymous fingerprinting," in *International Conference on the Theory and Applications of Cryptographic Techniques*, pp. 150–164, Springer, 1999.

[77] B. Pfitzmann and M. Schunter, "Asymmetric fingerprinting," in *International Conference on the Theory and Applications of Cryptographic Techniques*, pp. 84–95, Springer, 1996.

[78] Y. Yacobi, "Improved boneh-shaw content fingerprinting," in *Cryptographers' Track at the RSA Conference*, pp. 378–391, Springer, 2001.

[79] D. Dua and C. Graff, "UCI machine learning repository," 2017.

[80] G. Blakley, C. Meadows, and G. B. Purdy, "Fingerprinting long forgiving messages," in *Conference on the Theory and Application of Cryptographic Techniques*, pp. 180–189, Springer, 1985.