Author
Sebastian Gruber, B.Sc.

Submission
Department of Business Informatics – Data & Knowledge Engineering

Thesis Supervisor
o. Univ.-Prof. Dipl.-Ing. Dr. techn. Michael Schrefl

Assistant Thesis Supervisor
Mag. Dr. Bernd Neumayr

September 2020

Digital Watermarking of Medical Sensor Data for Data Leakage Detection – a Proof-of-Concept Prototype

Master Thesis

to obtain the academic degree of

Master of Science

in the Master's Program

Business Informatics

JOHANNES KEPLER UNIVERSITY LINZ
Altenberger Str. 69
4040 Linz, Austria
www.jku.at
DVR 0093696

JOHANNES KEPLER
UNIVERSITY LINZ

SWORN DECLARATION

Linz, 28.09.2020

Place, Date Signature

Acknowledgements

I would like to thank Prof. Michael Schrefl for giving me the chance to accomplish my master thesis in the department of Business Informatics – Data & Knowledge Engineering at Johannes Kepler University Linz. Furthermore, I would like to thank Dr. Christoph Fabianek and Dr. Eduard Gringinger who provided inspiration for the topic of my master thesis through the My Personal Connected Health (MyPCH) project. My greatest thanks go to Dr. Bernd Neumayr for his great advice and support in the course of this thesis.

Kurzfassung

Medizinische Geräte erzeugen medizinische Sensordaten, die zeitindizierte Sequenzen medizinischer Messungen sind. Mit Hilfe einer Datenplattform können Patienten ihre medizinischen Sensordaten dauerhaft speichern und sie an Datennutzer, wie Ärzten oder Forschern, weitergeben. Allerdings können Datennutzer Daten, auf die sie Zugriff haben, an Dritte weitergeben. Digitale Wasserzeichen können verwendet werden, um angeforderte Daten unmerklich mit einem Wasserzeichen zu versehen, welches dem Datennutzer, der die Daten angefordert hat, zugeordnet ist. Falls irgendwo Daten eines Patienten gefunden werden, kann der für die Weitergabe verantwortliche Datennutzer durch Extrahieren des eingebetteten Wasserzeichens identifiziert werden. In dieser Arbeit stellen wir einen neuen Wasserzeichen-Ansatz für medizinische Sensordaten zur Erkennung von illegaler Datenweitergabe vor. Da medizinische Sensordaten dauerhaft in der Datenplattform gespeichert werden, sind in diesem Wasserzeichen-Ansatz Einbettung und Erkennung von Wasserzeichen informiert, d.h. sie nutzen die Originaldaten. Darüber hinaus ist die Einbettung von Wasserzeichen durch Nutzungseinschränkungen konfigurierbar, dass Daten, die mit einem Wasserzeichen versehen werden, für diagnostische Zwecke nutzbar bleiben. Eine wesentliche Herausforderung für die Erkennung von Wasserzeichen sind böswillige Angriffe, die darauf abzielen, eingebettete Wasserzeichen zu verzerren oder zu entfernen. Daher basiert die Erkennung von Wasserzeichen auf Ähnlichkeitssuchen. In dieser Arbeit entwerfen, implementieren und evaluieren wir einen Prototyp, der auf dem eingeführten Wasserzeichen-Ansatz basiert. Die Evaluierung zeigt unter anderem, dass der Prototyp als sicher gegenüber gebräuchlichen Varianten von Verzerrungs-, Kollusions-, Teilmengenauswahl- und Löschangriffen angesehen werden kann. Darüber hinaus zeigt die Evaluierung, dass die Leistung des Prototyps als ausreichend für den praktischen Einsatz zur Bereitstellung kontinuierlicher Blutzuckermessungen für Datennutzer wie Ärzte angesehen werden kann. Der Prototyp kann auch auf verschiedene Arten von Sensordaten angewendet werden und bietet eine hohe Anpassungsfähigkeit sowie Erweiterbarkeit.

Abstract

Medical devices produce medical sensor data which are time indexed sequences of medical measurements. By using a data platform, patients can permanently store their medical sensor data and share it with authorized data users such as doctors or researchers. But authorized data users may leak accessed data to unauthorized parties. Digital watermarking can be used to imperceptibly watermark requested data with a watermark associated to the requesting data user. If leaked data is found anywhere, the leaking data user can be identified by extracting the embedded watermark. In this thesis, we introduce a new watermarking approach for medical sensor data to detect data leakage. Due to medical sensor data being permanently stored in the data platform, watermark embedding and detection are informed, i.e. they take advantage of original data. In addition, watermark embedding is configurable by usability constraints such that watermarked data remains useful for diagnostic purposes. A major challenge for watermark detection are malicious attacks which aim to distort or remove embedded watermarks. In order to counteract malicious attacks, watermark detection is based on similarity searches. In this thesis, we design, implement and evaluate a proof-of-concept prototype which is based on the introduced watermarking approach. Among other things, the evaluation shows that the prototype can be considered secure against common variants of distortion, collusion, subset selection and deletion attacks. Furthermore, the evaluation shows that the performance of the prototype can be considered sufficient for practical use of providing continuous blood glucose measurements for data users such as doctors. The prototype may also be applied to different kinds of sensor data and provides high adaptability as well as extensibility.

Table of Contents

1. Introduction

This chapter introduces basic concepts of this thesis and explains the problem statement as well as the solution idea. Furthermore, the goal of this thesis is defined, and an outline of the remaining chapters is provided.

1.1. Basic Concepts

This section introduces basic concepts of this thesis, namely medical sensor data, data leakage detection and digital watermarking.

Medical Sensor Data. Wireless sensor networks can be used in healthcare applications to monitor patients [1]. These networks consist of sensors which measure physical or environmental conditions and transmit their data to a central location [1]. In this thesis, medical devices of patients produce medical sensor data. A medical device consists of one or multiple sensors measuring medical data of a certain type and unit in periodic time intervals. A single measurement consists of a measured value together with its timestamp. Medical sensor data in this sense are time indexed sequences of measurements that may be regarded as time series data. Figure 1 illustrates the conceptual schema of medical sensor data.

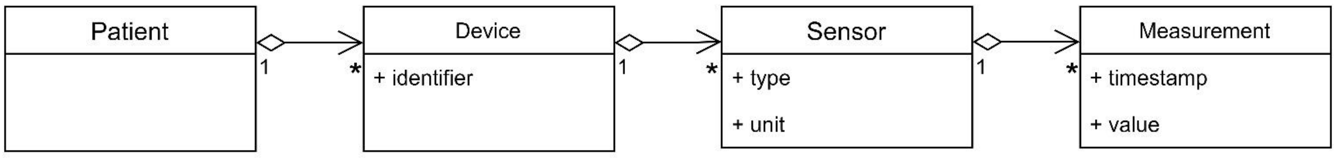


Figure 1: Conceptual Schema of Medical Sensor Data

Data Leakage Detection. Data leakage can occur, if sensitive data is transferred to supposedly trustworthy third parties [2]. The goal of data leakage detection is to detect when a supposedly trustworthy third party leaked data and to identify the data leaker [2]. In this thesis, it is assumed that a data platform manages medical sensor data of persons with diabetes (also referred to as patients). Patients can permanently store their data in the database of the data platform and provide access to authorized data users. In this case, sensitive data is transferred to supposedly trustworthy third parties which may result in data leakage. Typically, watermarking is used to handle data leakage detection [2].

Digital Watermarking. According to Cox et al. [3], watermarking can be defined as "… *the practice of imperceptibly altering a work to embed a message about that work*" [3]. There are several applications for watermarking such as transaction tracking (often also called "fingerprinting") [3]. In transaction tracking, watermarks are used to identify data users who legally access data but illegally redistribute it [3]. In this thesis, we use digital watermarking for transaction tracking (or fingerprinting) to detect data leakage of medical sensor data.

Watermark embedding and detection are informed due to medical sensor data being permanently stored in the database of the data platform. Watermark embedding can be much more effective if the to-be watermarked data is actually analyzed before encoding a watermark [3]. This is what we refer to as informed watermark embedding. Watermark detection can subtract original data from watermarked data to receive the independent and possibly noisy watermark [3]. This is what we refer to as informed watermark detection.

Malicious attacks are any processes which aim to defeat the purpose of the watermark [3]. Malicious attacks can prevent detection of watermarks, embed illegal watermarks, detect watermarks or attack the watermarking system [3]. Being aware of the presence of watermarks, malicious data users may attack data before leaking it. In order to counteract malicious attacks, we need a secure watermark on the one hand, and on the other hand, watermark detection must be able to take into account modifications of data and their embedded watermarks.

1.2. Problem Statement

After introducing basic concepts of this thesis, we present the problem statement which consists of assumptions regarding medical sensor data and the problem scenario with a data platform for storing and sharing medical sensor data.

Example Medical Sensor Data. In this thesis, medical sensor data are based on a published example dataset by Tidepool[1], a nonprofit organization providing a diabetes data hub. The published example dataset contains anonymized diabetes data from a demo user. For the sake of simplicity, we only consider measurements of continuous blood glucose. The resulting 59 454 measurements are adapted to provide a more general applicability by reducing them to a few important attributes only.
The following assumptions are made regarding the remaining attributes of continuous blood glucose measurements: “deviceId” contains the unique identifier of the medical device; “type” contains the data type measured by the sensor; “unit” contains the unit measured by the sensor; “time” contains the measurement’s timestamp; “value” contains the measurement’s value. Figure 2 shows example measurements in the JSON format from the example medical sensor data.

```
{
      "deviceId": "DexG5MobRec_SM64305440",
      "type": "cbg",
      "unit": "mmol/L",
      "time": "2017-02-04T16:45:23.000Z",
      "value": 6.993942468717372
},
{
      "deviceId": "DexG5MobRec_SM64305440",
      "time": "2017-02-04T16:50:23.000Z",
      "type": "cbg",
      "unit": "mmol/L",
      "value": 6.8274200289860065
}
```

Figure 2: Example Measurements in JSON format based on the example dataset from Tidepool[1]

In the example medical sensor data, measurements of continuous blood glucose are typically but not always performed every 5 minutes. This results in a maximum of 288 measurements for each day of each sensor measuring continuous blood glucose in mmol/L. The combination of measurements, based on their timestamps and sensor, results in a curve representing the evolution of the medical value which may be used for diagnostic purposes. Figure 3 visualizes an example combination of continuous blood glucose measurements in mmol/L from 2017-02-04.

[1] Tidepool: https://www.tidepool.org/

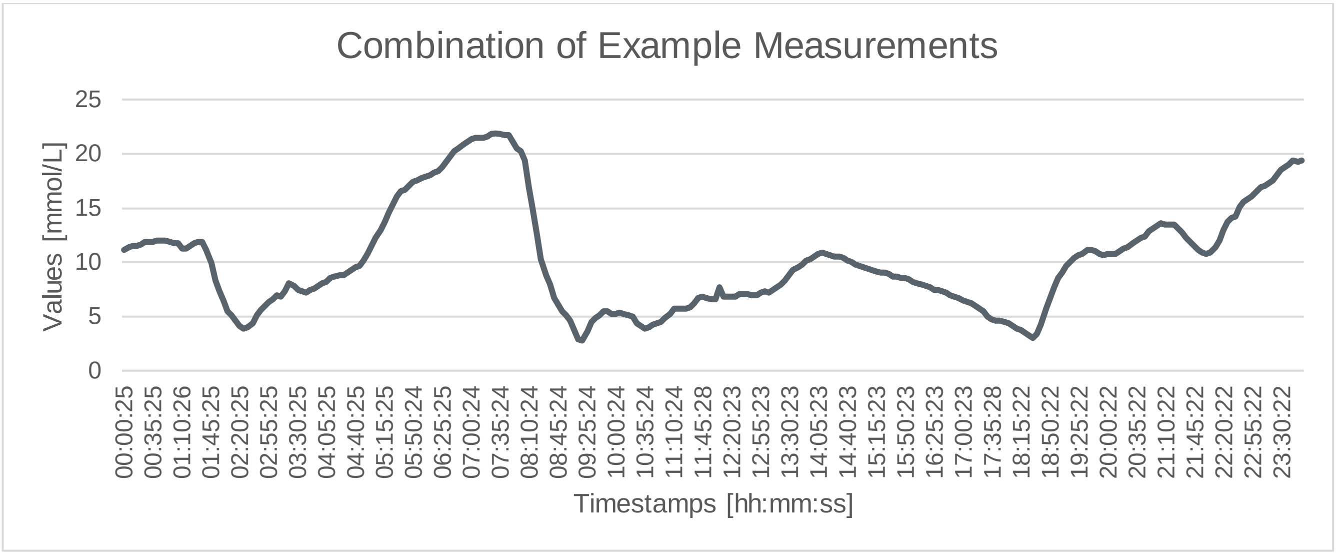


Figure 3: Combination of Example Measurements based on the example dataset from Tidepool[2]

Problem Scenario. In this thesis, the problem scenario is about a data provider who supplies a data platform which enables patients to efficiently manage their diabetes data. Patients can permanently store their medical sensor data in the database by using the container service. Once the data is stored in the database, it is not modified anymore. Patients can share their data by authorizing data users such as doctors or researchers. Authorized data users can then request data of patients by using the data service. Different data users may have different demands and usages of provided data. Doctors, as an example, may request data of a single patient of a day, week or month. In contrast to doctors, researchers may request data of many patients of a longer period of time. The data service retrieves requested data from the database and transmits it to the requesting data user. If an authorized data user illegally provides data to an unauthorized party, data leakage occurs. Figure 4 illustrates the problem scenario in which data leakage cannot be handled.

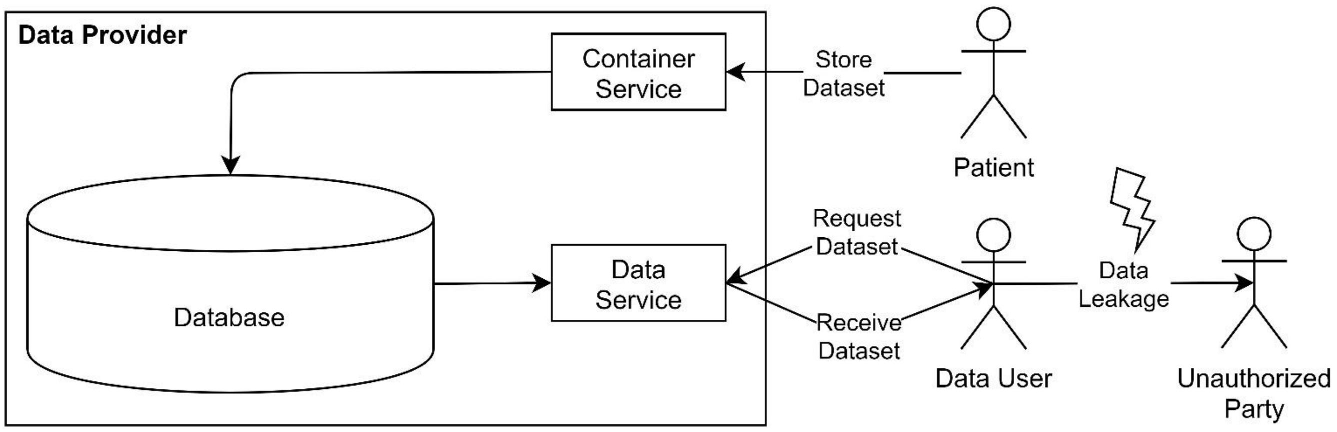


Figure 4: Problem Scenario with Data Leakage

It is physically impossible to prevent data leakage [4] but at least data leakage may be discouraged by watermarking requested data [5]. Nevertheless, watermarking can be used to identify leaking data users [2]. In conclusion, a watermarking approach to handle data leakage detection of medical sensor data is required.

[2] Tidepool: https://www.tidepool.org/

1.3. Solution Idea

This section introduces a new watermarking approach for data leakage detection of medical sensor data. The watermarking approach is the basis for the proof-of-concept prototype which is designed, implemented and evaluated in this thesis. We first give an overview of the watermarking approach and then explain dataset fragments, watermark embedding and watermark detection.

Overview. We adapt the problem scenario by adding dataset fragments, watermark embedding and watermark detection. First, dataset fragments are stored in the database to enable individual watermarking of small data units. Second, the data service additionally performs watermark embedding to watermark requested datasets with watermarks associated to its requesting data users. Third, the detection service is added to detect whether a suspicious dataset originates from the database and, if so, who leaked the data. Figure 5 illustrates the solution idea in which data leakage can be handled by extending the problem scenario of Figure 4 with dataset fragments, watermark embedding and watermark detection.

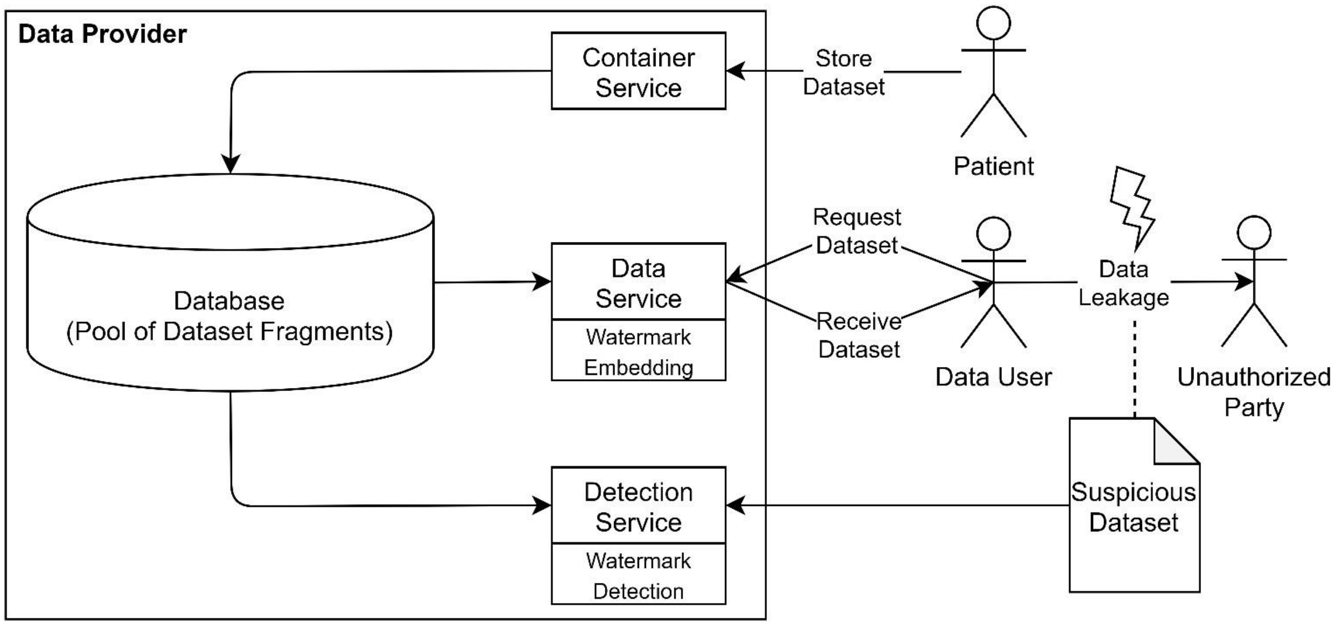


Figure 5: Solution Idea with Handling of Data Leakage

Dataset Fragments. Dataset fragments, or simply fragments, are small units of data. We assume that datasets can be stored in the database as fragments to enable individual watermarking of small units of data. The container service fragments a dataset by some fragmentation criteria resulting in a set of dataset fragments, see Figure 6. Instead of storing datasets, their corresponding sets of dataset fragments are stored in the database.

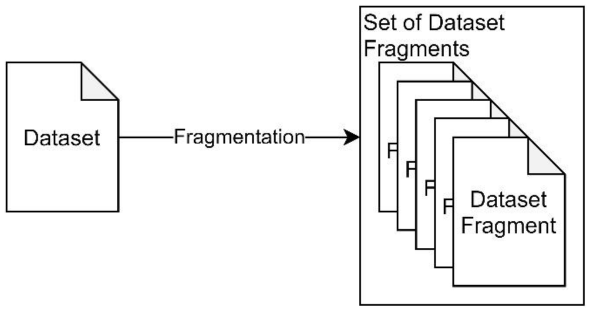


Figure 6: Fragmentation of a Dataset results in a corresponding Set of Dataset Fragments

The fragmentation criteria are based on the two dimensions sensor and time. The sensor dimension indicates that measurements of the same device, type and unit belong together. The time dimension indicates that measurements of a certain time period belong together. The time period is based on the smallest unit of usage of a measurement sequence. In this thesis, we assume that doctors typically request data of continuous blood glucose of a day, week or month, therefore the smallest unit of usage is a single day. Based on this assumption, datasets are fragmented per sensor and day. Figure 7 provides metadata of some example fragments of continuous blood glucose which are fragmented based on sensor and day. It should be mentioned that the previously shown Figure 3 does not visualize a random combination of example measurements but visualizes an example dataset fragment of continuous blood glucose in mmol/L from 2017-02-04.

```
Dataset Fragment 1<<Device X, Continuous Blood Glucose, mmol/L>, 2020-01-01>
Dataset Fragment 2<<Device X, Continuous Blood Glucose, mmol/L>, 2020-01-02>
Dataset Fragment 3<<Device Y, Continuous Blood Glucose, mmol/L>, 2020-01-02>
```

Figure 7: Example Metadata of Dataset Fragments of Continuous Blood Glucose

Watermark Embedding. Whenever an authorized data user requests data, the data service performs watermark embedding. A requested dataset corresponds to a set of dataset fragments. For each requested fragment, a matching watermark associated with its requesting data user is generated and embedded resulting in a watermarked fragment. The combination of all watermarked fragments results in the watermarked dataset which is transmitted to the requesting data user. The process of watermark embedding is illustrated in Figure 8 and discussed in more detail below.

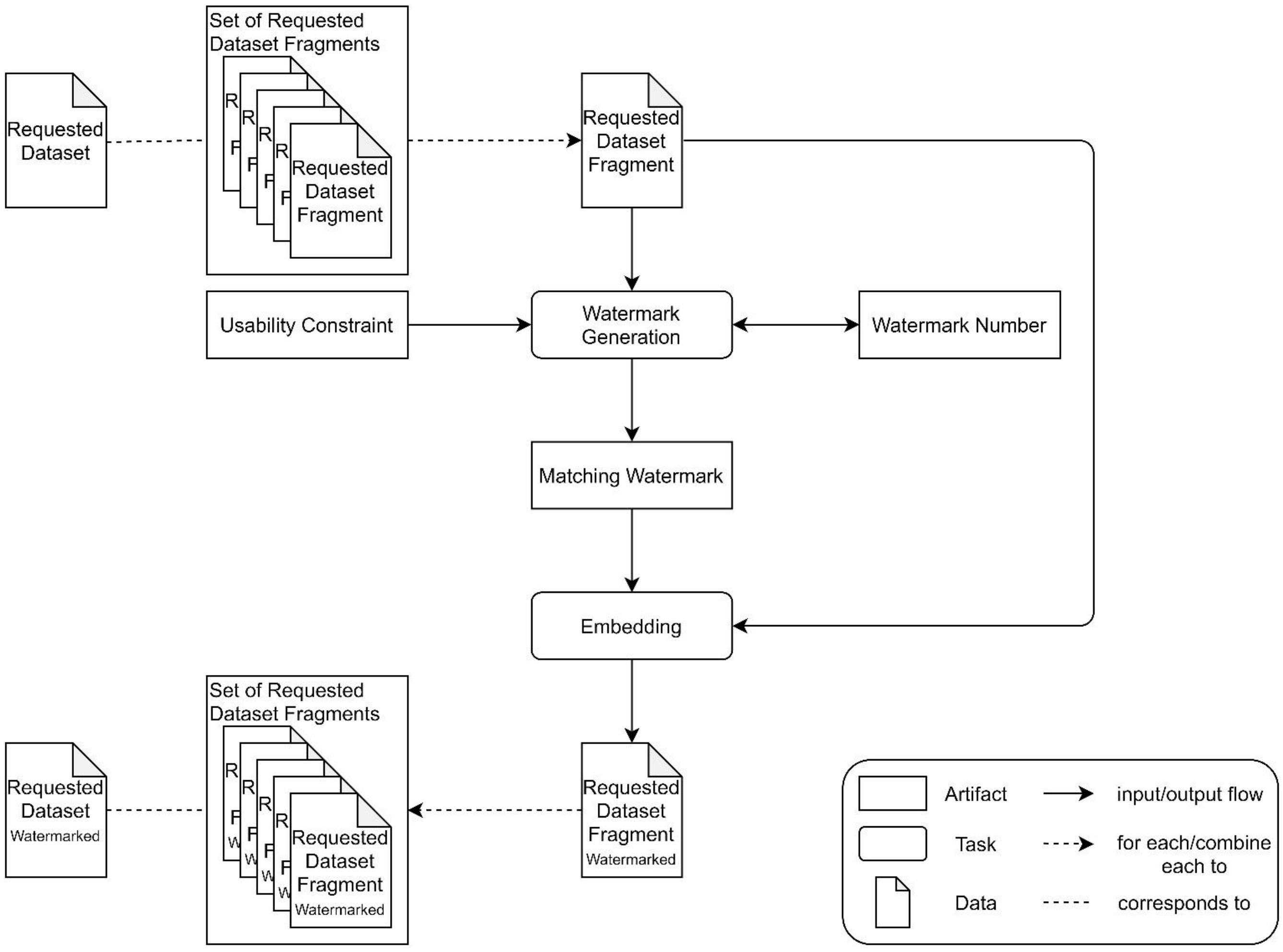

Requested Dataset
Set of Requested Dataset Fragments
Requested Dataset Fragment
Requested Dataset Fragment
Usability Constraint
Watermark Generation
Watermark Number
Matching Watermark
Embedding
Requested Dataset Watermarked
Set of Requested Dataset Fragments
Requested Dataset Fragment Watermarked
Requested Dataset Fragment Watermarked
Artifact
input/output flow
Task
for each/combine each to
Data
corresponds to

Figure 8: Process of Watermark Embedding

Watermark Generation. For each requested fragment, watermark generation produces a matching watermark based on the requested fragment, its usability constraint and a watermark number (based on previous requests). Ideally, watermark generation produces maximally different and very well matching watermarks. Maximally different watermarks enable detection of original watermarks more easily and more reliably even if extracted watermarks are noisy, only partially available or are the combination of multiple watermarks. Very well matching watermarks ensure very high imperceptibility, while preserving diagnostic meaning of data. The design of an ideal watermark generation, however, is out of scope of this thesis. We introduce a deterministic watermark generation to produce (not maximally) different and (not very well) matching watermarks. In the following, the parameters of watermark generation and their use are explained.

Requested Fragment. The watermark generation analyzes the requested fragment to produce a matching watermark. More precisely, watermark generation analyzes the measurement values of a requested fragment to mostly preserve the value structure such that the imperceptibility of watermarks is improved. For each measurement being watermarked, the value of the previously watermarked measurement $v_w(t-1)$, the value of the measurement being watermarked $v(t)$ and the value of the next measurement $v(t+1)$ are considered. Let α and β be comparison operators, $\alpha \in \{<, >\}$ and $\beta \in \{<, >\}$, the goal of watermark generation is to preserve the value structure, i.e. $v_w(t-1)\ \alpha\ v(t)\ \beta\ v(t+1) \rightarrow v_w(t-1)\ \alpha\ v_w(t)\ \beta\ v_w(t+1)$. There are two cases in which preserving the value structure is not desired, namely, if $v_w(t-1) = v(t)$ and if $v(t) = v(t+1)$.

The watermark generation also uses the requested fragment to securely produce a watermark. More precisely, watermark generation is secured by a secret (watermark) key to ensure security even if the algorithm is publicly known, see [3]. Any procedure providing a secret key may be used. For this purpose, we associate each dataset fragment with a random number as secret key with which it is stored in the database.

Usability Constraint. The watermark generation is limited by usability constraints to preserve metrics, such as standardized metrics for continuous glucose monitoring by Battelino et al. [6]. We assume that despite watermarking preserving important metrics of the data implies preserving diagnostic semantics. The usability constraint is defined by the data provider for each sensor, i.e. type and unit of measurements. Figure 9 illustrates usability constraints by extending the conceptual schema of medical sensor data from Figure 1.

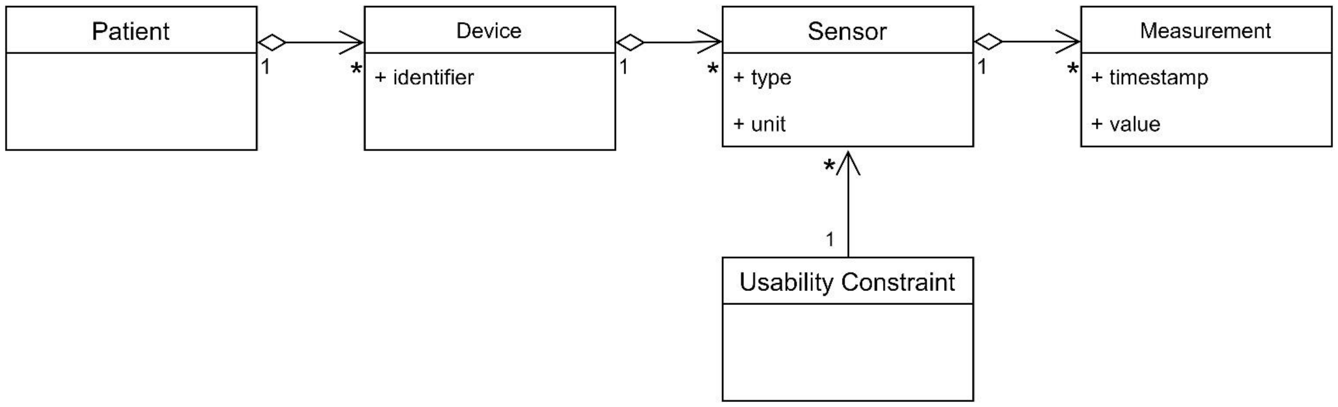


Figure 9: Conceptual Schema of Medical Sensor Data extended with Usability Constraints

In this thesis, we realize the usability constraint as a concept consisting of a maximum error, a sensor's minimum and maximum value as well as the number of ranges (explained below). The maximum error

determines the (initial) maximum error range for each measurement value. The maximum error range of each measurement is then possibly limited by its value structure (explained above) and by the sensor's minimum and maximum values. In the end, the number of ranges configures the error distribution by indicating the number of error sub-ranges within the possible error range with each error sub-range having a probability with which it is selected.
The error sub-ranges and their selection probabilities are determined as follows. The maximum error range is divided into a lower error range, which is below the original value, and an upper error range, which is above the original value. The error sub-ranges indicated by the number of ranges are halved, with one half being distributed to the lower error range and the other half to the upper error range. The closer an error sub-range is to the original value, the higher the probability to be selected. In other words, the higher the number of ranges, the closer the errors may be distributed to the original value. Table 1 provides some example selection probabilities of error sub-ranges using various configurations regarding the number of ranges.

Number of Ranges	Selection Probabilities of Error Sub-Ranges of the Lower Error Range				Selection Probabilities of Error Sub-Ranges of the Upper Error Range				Total
	4	3	2	1	1	2	3	4	
2				50.0 %	50.0 %				100 %
4			25.0 %	25.0 %	25.0 %	25.0 %			100 %
6		12.5 %	12.5 %	25.0 %	25.0 %	12.5 %	12.5 %		100 %
8	6.25 %	6.25 %	12.5 %	25.0 %	25.0 %	12.5 %	6.25 %	6.25 %	100 %

Table 1: Example Selection Probabilities of Error Sub-Ranges using different Number of Ranges

Let us provide an example for applying such a usability constraint. In this example, assume a measurement with an original value of 1.0 and a usability constraint with a maximum error of 0.6, a sensor's minimum value of 0.6, a sensor's maximum value of 2.0 and a number of ranges of 8.
Based on the maximum error, the initial maximum error range is [0.4, 1.6]. The sensor's minimum value limits the error range, while the sensor's maximum value does not limit the error range. The resulting possible error range is [0.6, 1.6]. For the sake of simplicity, we assume that the value structure of measurement values does not limit the error range. Now, the number of ranges are applied by introducing error sub-ranges to the lower error range of [0.6, 1.0] and the upper error range of [1.0, 1.6]. The resulting error sub-ranges together with their selection probabilities are shown in Table 2.

Lower Error Sub-Ranges		Upper Error Sub-Ranges	
Error Sub-Range	Selection Probability	Error Sub-Range	Selection Probability
[0.6, 0.7]	6.25 %	[1.00, 1.15]	25.00 %
[0.7, 0.8]	6.25 %	[1.15, 1.30]	12.50 %
[0.8, 0.9]	12.50 %	[1.30, 1.45]	6.25 %
[0.9, 1.0]	25.00 %	[1.45, 1.60]	6.25 %

Table 2: Example Error Sub-Ranges together with their Selection Probabilities

In summary, a tight usability constraint, i.e. small maximum error and high number of ranges, results in small errors providing high imperceptibility but less security. A loose usability constraint, i.e. high maximum error and low number of ranges, results in high errors providing high security but less imperceptibility. The selection of an appropriate usability constraint is one of the major challenges for data providers using such a usability constraint.

Watermark Number. The watermark generation determines the watermark to-be produced based on a watermark number. A watermark number is associated with a requesting data user and a requested fragment. In other words, a watermark corresponds to a watermark number which together with the fragment identifies a data user. A request log keeps track of already embedded watermarks by storing requesting data users, requested fragments and associated watermark numbers. If a data user requests a fragment for the first time, it is associated with a watermark number which is stored in the request log. For every additional request of the same fragment from the same data user, the same watermark is embedded by looking up the watermark number in the request log such that a single data user cannot receive multiple differently watermarked copies of the same data.

Matching Watermark. The result of watermark generation is a matching watermark which is a sequence of matching numeric errors with the sequence having the same length as the requested fragment, i.e. the number of errors is the same as the number of measurements. The watermark consists of numeric errors, because it is embedded only in the measurement values of a requested fragment. Embedding of errors in the fields “deviceId”, “type” or “unit” is not safe due to malicious attackers can easily detect and remove such errors. Furthermore, embedding of errors in the “time” fields is not secure, because a measurement sequence without the “time” fields still represents meaningful information if the measuring time intervals are close to equal.

The watermark has the same length as the requested fragment because the watermark is embedded in all measurements of the requested fragment to provide high security especially against colluding attackers. In addition, even if parts of a fragment are removed, parts of the watermark are still present. Figure 10 visualizes an example watermark with the same length as the number of measurements in the dataset fragment from Figure 3. The watermark introduces errors to every measurement value of the fragment. The watermark was generated using a usability constraint with a maximum error of 0.5 and a number of ranges of 10.

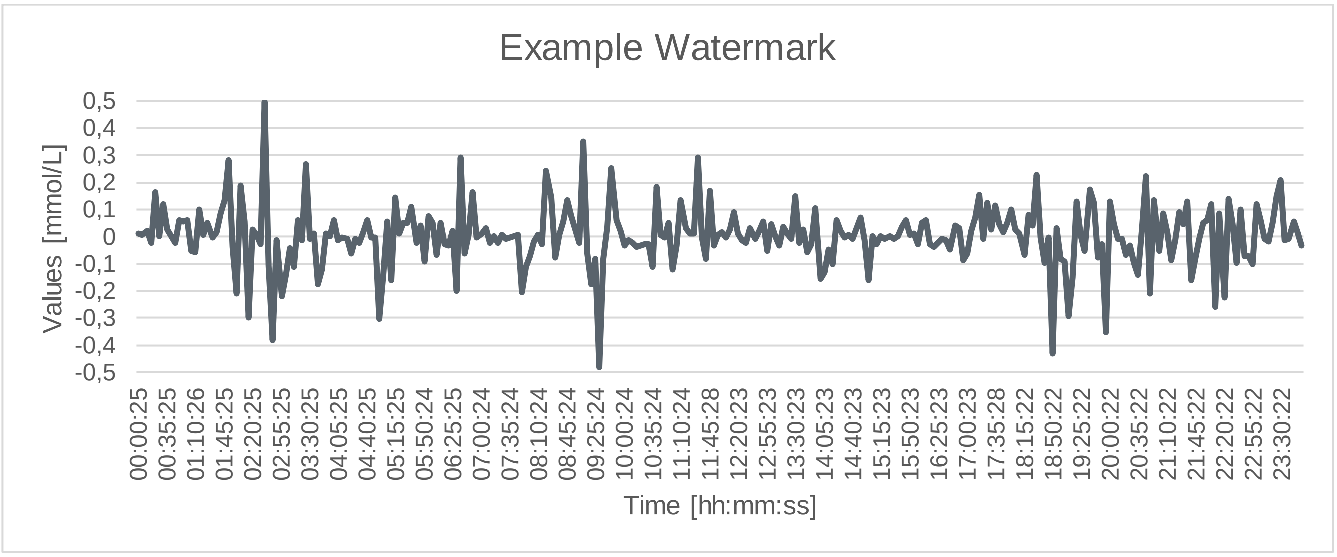


Figure 10: Example Watermark for the Dataset Fragment from Figure 3

Embedding. After the watermark generation produced a matching watermark, the watermark can be embedded into the requested fragment which results in a watermarked fragment. With a watermark being a sequence of matching errors with the same length as the requested fragment, it can be embedded by adding each error to the corresponding measurement value. Figure 11 visualizes an example watermarked fragment by embedding the watermark from Figure 10 into the dataset fragment from Figure 3.

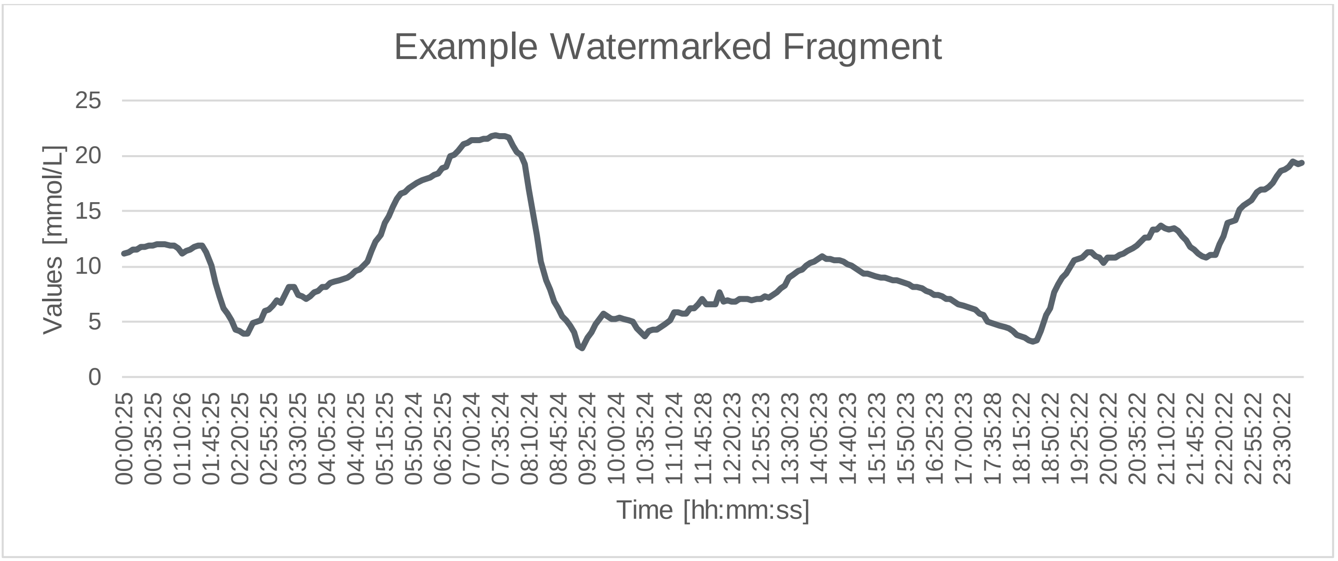
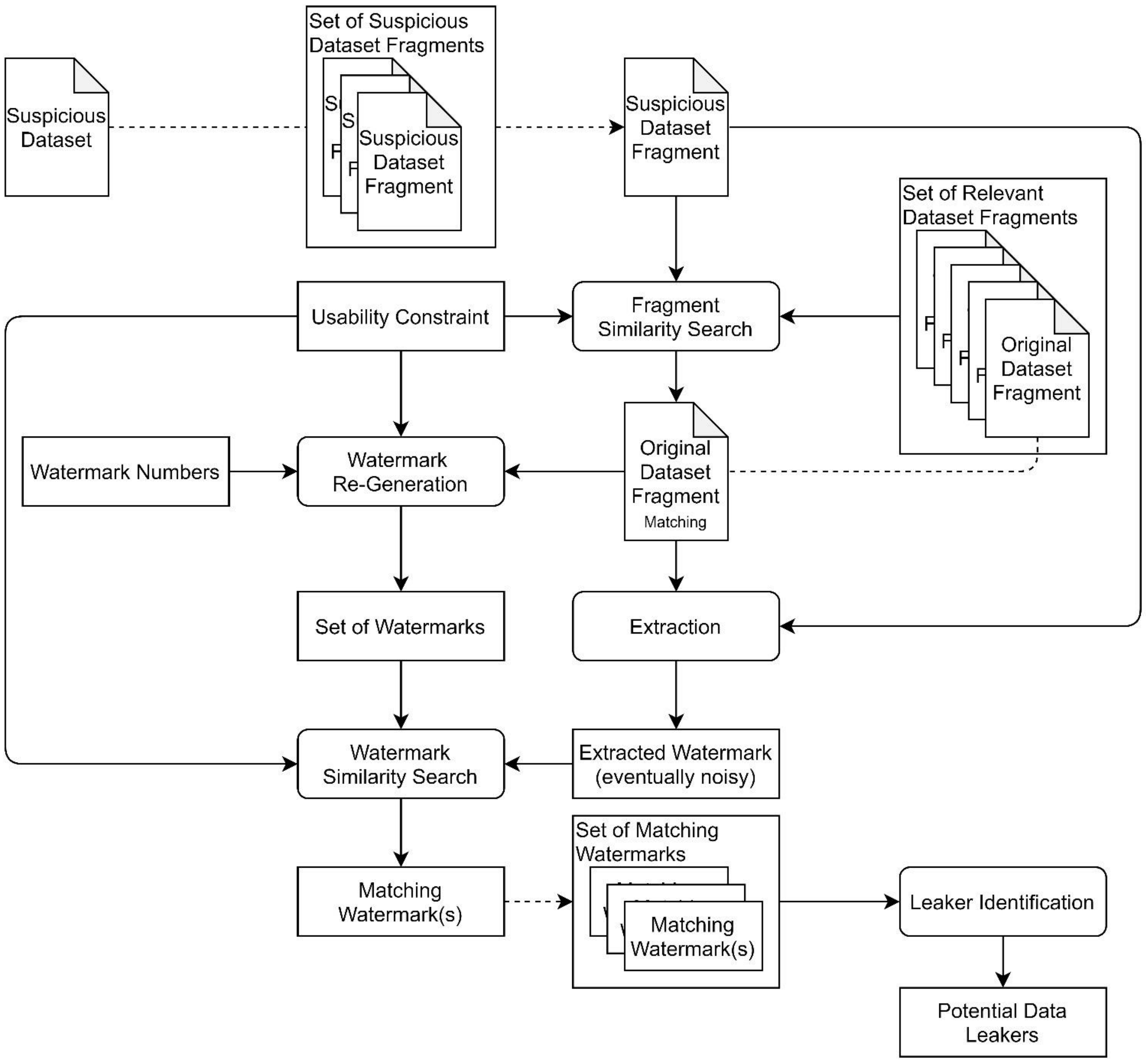


Figure 11: Example Watermarked Fragment by combining Figure 3 and Figure 10



Suspicious Dataset
Set of Suspicious Dataset Fragments
Suspicious Dataset Fragment
Suspicious Dataset Fragment
Set of Relevant Dataset Fragments
Original Dataset Fragment
Usability Constraint
Fragment Similarity Search
Watermark Numbers
Watermark Re-Generation
Original Dataset Fragment Matching
Set of Watermarks
Extraction
Watermark Similarity Search
Extracted Watermark (eventually noisy)
Set of Matching Watermarks
Matching Watermark(s)
Matching Watermark(s)
Leaker Identification
Potential Data Leakers

Figure 12: Process of Watermark Detection

Watermark Detection. Whenever a suspicious dataset is found somewhere, the detection service can perform watermark detection to determine whether a suspicious dataset originates from the database and, if so, who leaked the data. A suspicious dataset corresponds to a set of suspicious dataset fragments. For each suspicious fragment, a matching original fragment is identified by using fragment similarity search. If a matching fragment is found, the embedded watermark is extracted, and already embedded watermarks of the matching fragment are re-generated. The extracted watermark may be noisy due to malicious attacks. To this end, matching watermarks are identified with watermark similarity search. The combination of matching watermarks of a suspicious dataset can reliably identify a leaking data user. The process of watermark detection is illustrated in Figure 12 and discussed in more detail below.

Fragment Similarity Search. The fragment similarity search identifies the matching fragment of a suspicious fragment based on original fragments. Especially for fragment similarity search, we take advantage of medical sensor data always being available. Only relevant original fragments are retrieved from the database to limit the number of fragments for fragment similarity search. The fragment similarity search computes fragment similarities between a suspicious fragment and each fragment from the set of relevant fragments based on their matching measurements. If an original fragment with the highest similarity is above a user-defined threshold, it is assumed that the matching fragment is found.

The problem of fragment similarity search is to identify matching measurements between a suspicious fragment and an original fragment due to modifications the number and position of comparable measurements may differ. The matching measurements analysis therefore identifies comparable measurements between two fragments. Table 3 justifies a matching measurements analysis by listing measurements of an original fragment and three suspicious (modified) fragments of this fragment. If the original fragment and the suspicious fragment have the same length, measurements can simply be compared sequentially. If the number or position of measurements differs, the similarity cannot simply be computed sequentially.
We assume that measurements match if they have an equal timestamp. This assumption may improve detection because matching measurements are found easily and reliably. Nevertheless, this assumption also results in a vulnerability to malicious attacks targeting timestamps of measurements, e.g. time shifting attacks. The design of an ideal fragment similarity search, however, is out of scope of this thesis.

Fragments	Measurements M_1, …, M_{10}									
Original	M_1	M_2	M_3	M_4	M_5	M_6	M_7	M_8	M_9	M_{10}
Suspicious 1	M_1	M_2	M_3	M_4	M_5	M_6	M_7	M_8	M_9	M_{10}
Suspicious 2	M_4	M_5	M_6	M_7						
Suspicious 3	M_1	M_3	M_5	M_7	M_9					

Table 3: Justification of Matching Measurements Analysis

Extraction. After the matching fragment of a suspicious fragment is identified, the embedded watermark is extracted. To this end, each measurement value of the matching fragment is subtracted from the corresponding measurement value of the suspicious fragment. The resulting sequence of differences (or errors) is the extracted and due to malicious attacks eventually noisy watermark. The extraction of a watermark by subtraction usually improves detection performance, see [3].

Watermark Re-Generation. The already embedded watermarks of the matching fragment can be re-generated based on previous requests. For this purpose, the already embedded watermark numbers of the matching fragment are retrieved from the request log. The watermark numbers are then used to re-generate a set of already embedded watermarks using the same algorithm as for watermark generation in watermark embedding. In addition, this set of re-generated watermarks may be extended by computing watermark combinations to improve security against colluding data users.

Watermark Similarity Search. The watermark similarity search identifies matching watermarks of an extracted watermark from the set of re-generated watermarks by computing watermark similarity. If a re-generated watermark has a watermark similarity above a user-defined threshold, it is considered as matching watermark. There may be multiple matching watermarks of an extracted watermark such that a single fragment may not reliably identify a leaking data user. In order to identify the leaking data user of a suspicious dataset, matching watermarks are stored in a set of matching watermarks.

Leaker Identification. After watermark similarity search was performed for each suspicious fragment, the leaker identification identifies the data leaker of a suspicious dataset based on the set of matching watermarks. The average watermark similarity of watermarks associated to a data user indicates its probability of being the data leaker of a suspicious dataset. In contrast to a single fragment which may not reliably identify the data leaker, multiple fragments reliably identify the data leaker by the watermark combination. The result of leaker identification are data users which are considered as potential data leakers together with their leakage probabilities.

1.4. Goal

The goal of this thesis is to implement the introduced watermarking approach for data leakage detection of medical sensor data in a proof-of-concept prototype. We determine requirements and design goals for the prototype and evaluate it on this basis.

1.5. Outline

Section 2. The state of the art of watermarking in general and watermarking of relational databases is introduced. In the end, the state of the art is discussed and related work from watermarking of time series data is investigated.

Section 3. The requirements and design goals for the proof-of-concept prototype are determined based on the context of this thesis. In addition, a summary of requirements and design goals together with their evaluation criteria is provided.

Section 4. The design of watermark embedding and detection for the proof-of-concept prototype are determined based on the watermarking approach introduced in the solution idea. Furthermore, expected results and limitations are discussed.

Section 5. The implementation of the proof-of-concept prototype is explained. In addition, the command line interface intended for experimental use is introduced.

Section 6. The proof-of-concept prototype is evaluated regarding the requirements and design goals from Section 3. In the end, the result of the evaluation is summarized.

Section 7. This thesis is summarized and concluded.

2. State of the Art

This chapter presents the state of the art of digital watermarking in general and watermarking of relational databases. Digital watermarking in general is introduced mostly based on the text book by Cox et al. [3]. Digital watermarking of relational databases, especially for fingerprinting, is introduced because medical sensor data could be stored in relational databases. In the course of this thesis, it turned out that watermarking of time series data may be more relevant than watermarking of relational databases. We therefore discuss the state of the art and investigate related work of watermarking of time series data.

2.1. Digital Watermarking

This section differentiates digital watermarking from cryptography and steganography. Furthermore, a general watermarking process is provided, and applications as well as properties of watermarking are listed.

2.1.1. Cryptography, Steganography and Watermarking

With the information growth of the last decades the need of securing content from theft emerged [7]. Therefore, algorithms and techniques including cryptography, steganography and watermarking are used to secure information [7]. Cryptography applies encryption and decryption to make information secret or unreadable [7]. Steganography hides secret information on carrier files such as images, audio etc. [7]. Cryptography consists of systems which make information unreadable or inaccessible, while steganography consists of systems which make information unnoticeable [8]. Digital watermarking embeds secret information in the digital content for security, integrity and authentication [7]. The digital watermark is inserted into a digital document and carries information such as the owner, creator or authorized consumer [9]. Watermarking is closely related to steganography but there are a few differences [9].

Cox et al. [3] define steganography as "*the practice of undetectably altering a work to embed a secret message*", while watermarking is defined as "*the practice of imperceptibly altering a work to embed a message about that work*" [3]. Comparing these two definitions, the differences between steganography and watermarking become clear. In contrast to steganography which undetectably alters an object, watermarking imperceptibly alters an object. Furthermore, steganography embeds any secret message, while in watermarking the message is about the object.

Digital watermarking started in 1979, gained popularity in 1990 and became fully applied in 1998 [9]. The internet became more user friendly and with the internet as distribution system, the content owner faced the risk of piracy [3]. Watermarking is applied to an object by inserting small errors (marks) into the object with all inserted errors together constituting the watermark [10]. The errors are insignificant for data usefulness and cannot be removed without significantly reducing data usefulness [10]. Watermarking is an alternative or complement to cryptography because it can protect content after decryption and survive processing of the object [3]. There are three important attributes which distinguish watermarking from other techniques [3]. Watermarks are *imperceptible* in objects, *inseparable* from objects and undergo the *same transformations* as the objects [3]. These three attributes make watermarking valuable for specific applications [3].

2.1.1. General Watermarking Process

Watermarking is a process of embedding a watermark into an object [11]. The generic watermarking system (see Figure 13) consists of an embedder and a detector [3]. An embedder has an object to-be watermarked and a watermark as input, and a watermarked object as output [3]. A watermarked object may be published in the web or sold to a customer, and is possibly manipulated after distribution [12]. A detector has any object as input, eventually determining if a watermark is present and with a detected watermark as output [3]. The detected watermark can be analyzed regarding its similarity to an original watermark [12].

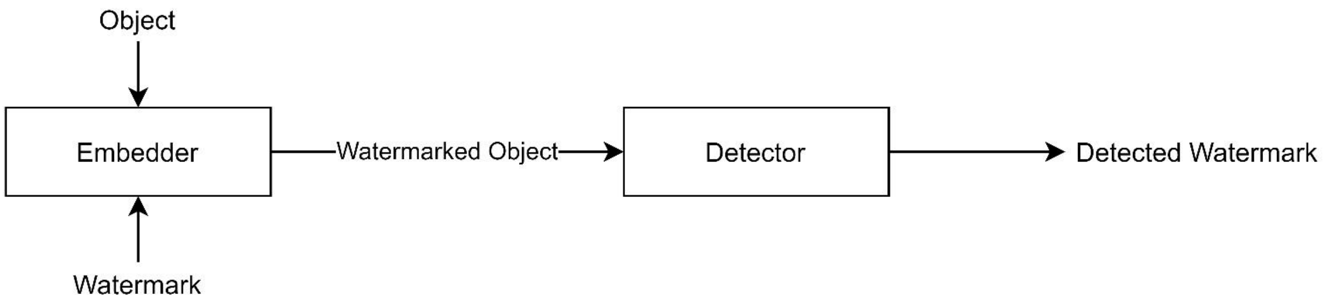


Figure 13: Generic Watermarking System, adapted from [3]

Watermark embedding and detection can be informed or blind with the difference being the use of the original object. The embedding process basically consists of two steps [3]. The first step maps a message *m* into an added pattern w_a, which has the cover object's type and dimension [3]. The second step adds w_a to the cover object c_0 resulting in the watermarked object c_w [3]. This is also called blind embedding. Informed embedding enables w_a to be dependent on c_0 by allowing the encoder to examine c_0 before encoding w_a [3].
After embedding, noise *n* may be introduced to c_w by processing, e.g. malicious attacks, resulting in c_{wn} [3]. The detection also consists of two steps [3]. The first step subtracts c_0 from c_{wn} resulting in a noisy watermark pattern w_n [3]. The second step uses the watermark decoder to decode w_n with the watermark key to receive the noisy message m_n [3]. This is also called informed detection [3]. Blind detection does not use the cover object c_0 for detection, thus, it cannot be removed before decoding [3].

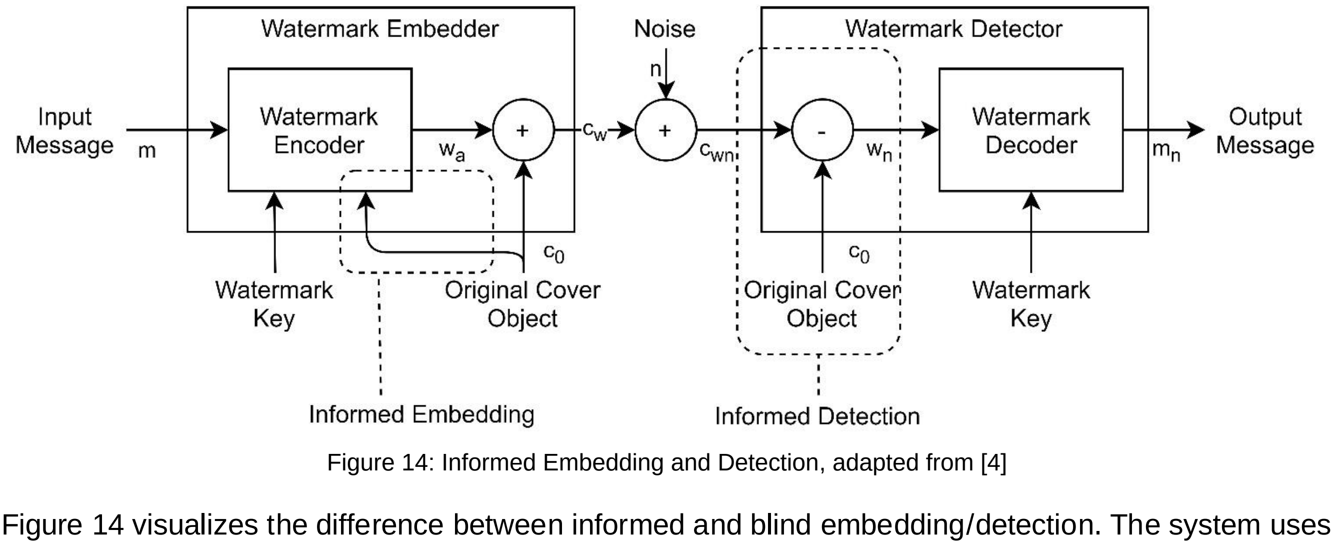


Figure 14: Informed Embedding and Detection, adapted from [4]

Figure 14 visualizes the difference between informed and blind embedding/detection. The system uses informed embedding/detection if the process within the dotted lines is considered as part of the system. The system uses blind embedding/detection if the process within the dotted lines is not considered as part of the system.

2.1.1. Watermarking Applications

Watermarking has various applications which are listed in the following paragraph. In *broadcast monitoring*, a unique watermark is embedded into each object before broadcasting [13]. The broadcast is monitored and based on embedded watermarks, it can be identified when and where objects appear [3, 13]. In *owner identification*, the copyright owner's identity is embedded into the object by watermarking [3]. The copyright notice may be removed from the object but with the watermark as an integral part of the object, the owner of the object can be identified [13]. In *proof of ownership*, owners not only want to identify ownership but also prove ownership [13]. The embedded watermark can provide evidence for ownership in disputes [3]. In *transaction tracking*, leakers, who obtain objects legally and redistribute them illegally, are identified by watermarking [3]. Therefore, a unique watermark is embedded into each individual copy of the same object [13]. Transaction tracking is also often called "fingerprinting", but this term also refers to human fingerprints and passive identification in broadcast monitoring [3]. In *content authentication*, the signature information can be embedded by watermarking to verify that the object has not been tampered [3]. Especially in legal cases and medical imaging, the integrity of an object is critical [13]. In *copy control*, recording of copyrighted objects can be prohibited based on the embedded watermark [3]. In *device control*, devices detect watermarks in objects in order to react to them, e.g. by adding value to the content [3]. In *legacy enhancements*, watermarks are used to improve the functionality of legacy systems [3].

Besides different applications, watermarking can also be applied to different data types. It can be applied, for example, to different media types such as text, image, video and audio [14, 15], software [10] or even to deep neural networks [16]. The increased remote usage of databases through the internet also created the need for watermarking of databases [10]. Furthermore, other data types such as graph structured data, spatial data or sequence data are also targets of watermarking [17]. In summary, there are different applications for watermarking and also different application areas. The individual application determines the required properties and appropriate evaluation criteria [3].

2.1.2. Watermarking Properties

The watermarking properties in this sub-section can be used to characterize watermarking systems [3]. The importance of a property depends on the application and on the role of the watermark [3]. Even the interpretation of a property depends on the application [3]. The following properties are discussed based on Cox et al. [3]: effectiveness, fidelity, payload, blind or informed detection, false positive behavior, robustness, security, secret keys, modifications and multiple watermarks as well as costs. It is impossible for a watermarking system to fulfill all properties and, therefore, tradeoffs dependent on the application must be made [13].

The *embedding effectiveness* is the probability of successfully embedding a watermark in an object or the probability of successful watermark detection after embedding [3]. Typically, 100% effectiveness is desired but often comes at a high cost regarding other properties [3]. Some effectiveness may be sacrificed for better performance which depends on the individual application [3].

The *fidelity* refers to the perceptual quality of the watermarked object, i.e. the similarity between an original and watermarked object [3]. Since the watermarked object may be degraded by the transmission, fidelity refers to the similarity at the point of presentation [3]. Mildly perceptible watermarks may be acceptable for higher robustness or lower costs which depends on the application [3]. The tradeoff between robustness and invisibility is an important issue for developing a robust but invisible watermark [14].

The *data payload* refers to the amount of information (bits) a watermark carries and depends on the application [3]. The data payload can be specified for a unit of time or within an object [3].

The watermark detector may not need additional information (*blind detection*) or needs additional information about the original object (*informed detection*) [3]. In transaction tracking applications, the original object is typically available which may improve detection performance because subtracting the original object from the watermarked object results in the watermark [3]. Furthermore, the original object may be used to counteract temporal or geometric distortions introduced to the watermarked object [3]. There are also applications which require the detection to be blind such as copy control [3].

The *false positive rate* refers to the frequency of falsely detected watermarks in unwatermarked objects [3]. The false positive probability can be defined as the probability to detect a watermark in a fixed object with randomly selected watermarks or to detect a watermark in randomly selected objects with a fixed watermark [3]. Typically, the second definition of false positive probability is more interesting except for a few cases such as transaction tracking, because false detection in this sense may result in false theft accusation [3].

The *robustness* refers to the ability of being able to detect a watermark in a (normal) processed watermarked object [3]. The robustness to common or normal processing operations depends on the application [3]. There are cases requiring robustness only against common processing operations or even no operations such as in fragile watermarking [3]. There are also cases where the watermark must be secure requiring robustness against all processing operations which do not destroy the value of the object [3].

The *security* refers to the ability of a watermark to withstand malicious attacks [3]. There are the following attack categories [3]: unauthorized removal, unauthorized embedding, unauthorized detection and system attacks. Unauthorized removal aims to prevent watermark detection by elimination or masking attacks [3]. An object attacked by elimination attack does not contain a watermark anymore, while the watermark of an object attacked by masking attack cannot be detected anymore [3]. The collusion attack is a form of unauthorized removal and especially a threat in transaction tracking [3]. An attacker combines multiple copies of the same object, containing different watermarks, to obtain a copy which contains no watermark at all [3]. Unauthorized embedding (forgery) aims to embed illegal watermarks into objects [3]. Unauthorized detection (passive attacks) aims to detect the embedded message, the embedded watermark or the presence of a watermark [3]. Finally, system attacks exploit the usage of watermarks but not the watermarks itself, i.e. the watermarking system is the target of attacks [3]. The prevention of one or more attack categories are necessary depending on the application [3].

Cipher keys refer to the use of secret keys for message encryption, while *watermark keys* refer to the use of secret keys for watermark embedding and detection [3]. Typically, security of an algorithm is based on secret keys and not by the algorithm itself [3]. It is often desired to use both key forms in watermarking systems [3].

Modifications refer to the ability of changing embedded watermarks, while *multiple watermarks* refer to the ability to embed multiple watermarks [3]. There are applications in which modifying the embedded watermark is not desired, but there are also applications which require modifications such as in copy control applications [3]. The alternative to modifying an embedded watermark is to embed another one as often required in transactional watermarking applications [3].

The *costs* refer to the computational costs regarding watermark embedding and detection [3]. Especially, the speed of embedding and detection as well as the number of embedders and detectors are interesting [3]. In contrast to broadcast monitoring which requires real time embedding and detection, in rare disputes of proofing ownership a detector provides value even if days are necessary to detect a watermark [3]. The number of embedders and detectors also depends on the application such as in transaction tracking where more embedders than detectors may be employed [3].

In this section, digital watermarking in general based on the text book by Cox et al. [3] was introduced to give a basic overview of the subject of watermarking. In the next section, digital watermarking of relational databases is introduced.

2.2. Digital Watermarking of Relational Databases

After introducing digital watermarking in general, digital watermarking of relational databases is introduced because medical sensor data could be stored in relational databases. This section provides the need for watermarking of relational databases as well as its applications, properties and possible attacks. Especially fingerprinting as an application of watermarking is investigated in more detail.

2.2.1. The Need of Watermarking of Relational Databases

The research of watermarking was focused mostly on images, audio, video etc. [5] resulting in a large amount of literature for watermarking of multimedia data [10]. The need for watermarking of relational databases emerged due to an increased use of databases in applications [10, 18]. The trend of enabling users to search and access databases remotely by the internet resulted in the threat of data theft for data providers [10]. Database watermarking may be important in different cases such as in selling data to parties specialized in data mining [5]. It is crucial to prevent data users from illegal database distribution [4]. Therefore, a technology was required to identify pirated database copies [10].

The techniques for watermarking of multimedia data cannot be directly used for watermarking of relational data because of the following differences [10]. First, a multimedia object consists of many redundant bits resulting in a large coverage for the watermark to hide, while a database relation consists of tuples with the watermark being spread over different objects [10]. Second, parts of a multimedia object does not change its spatial or temporal positioning, while there is no implied ordering between tuples in relational data [10]. Third, multimedia objects typically remain intact, while normal processing operations (insert, update, delete) are performed in databases [10]. Fourth, in multimedia objects psycho-physical phenomena may be used for watermark embedding which is not possible in relational databases [5]. Because of these differences between multimedia and relational data, techniques for watermarking of multimedia data cannot be directly used for watermarking of relational data [10].

Watermarking techniques of text and software are also not applicable to relational data [10]. The techniques for watermarking of text usually exploit properties of text formatting and semantics which might not be useful in the context of simple data types of relational databases [10]. The techniques for watermarking of software also hold only little promise, because rearranging the code without altering its semantics may destroy the watermark [10]. In summary, watermarking techniques of multimedia data, text and software are not directly applicable to relational data [10].

2.2.2. Relational Database Watermarking Applications

Watermarking of relational databases is useful for various applications especially ownership assertion (or copyright protection), fingerprinting (or transaction tracking) as well as fraud and tamper detection [5].

The aim of *copyright protection* is to prove ownership based on an embedded watermark [18]. Therefore, a database can be watermarked using private parameters and then be made publicly available [5]. If a suspect relation is published, ownership can be proven by showing the presence of the watermark [5].

The watermark must survive data processing operations which remove or distort the watermark [5]. Single and joint ownership of a database can be proven by a secret key [18]. In contrast to a single ownership in which only the owner has the key, in joint ownership all owners have the same key [18]. This may be problematic if any owner illegally distributes assets without the permission of other owners [18].

The aim of *fingerprinting* is to identify traitors [5, 18, 19]. An owner embeds a distinct fingerprint in each copy of the database to discourage unauthorized duplication and distribution [5]. If unauthorized copies of a database are found, the origin can be identified based on the embedded fingerprint [5]. In contrast to copyright protection which aims to prove ownership, fingerprinting aims to identify a traitor [19]. Furthermore, copyright protection is attacked in a way that an attacker claims ownership or weaken a merchant’s claim, while fingerprinting is attacked in a way such that an innocent or no principal is identified as traitor [19]. In the context of watermarking relational databases typically the term “fingerprinting” is rather used than “transaction tracking”. In the context of relational databases, we also use the term “fingerprinting”. Fingerprinting of relational databases is further investigated in Section 2.2.5.

The aim of *fraud and tamper detection* is to detect and localize or recover modifications based on a fragile watermark [18]. Especially in critical applications, it is important to ensure that the data and its source have not been changed, manipulated or modified [5]. Therefore, the integrity of data is verified by extracting the embedded watermark [5]. In contrast to robust watermarks, fragile watermarks are fragile to any modifications [5].

2.2.3. Relational Database Watermarking Properties

In the literature various properties, requirements or issues for watermarking systems of relational databases are proposed such as in [5, 10, 20–22]. In this thesis, we name properties for watermarking of relational databases based on the pioneer work for watermarking of relational data by Agrawal et al. [10].

The properties named by Agrawal et al. [10] are robustness, accuracy, incremental updatability as well as blind and public system. The *robustness* refers to the watermark being robust against degradations by benign updates or malicious attacks [10]. The *accuracy* refers to the watermark not being detected in any other non-pirated database, i.e. false hits [10]. The *incremental updatability* refers to the watermark being only recomputed for modified and added tuples [10]. The *blind system* refers to the watermark detection not requiring the watermark or original database [10]. The public system refers to the defense of a watermark system being based on the choice of a secret key assuming that the algorithm for watermark embedding is publicly known [10].

2.2.4. Relational Database Watermarking Attacks

The watermark may be damaged, erased or compromised by intentional and unintentional attacks [10]. The watermark must therefore be secure against attacks, but not hinder normal data processing operations [10].

Possible attacks include benign updates, malicious attacks and false claims of ownership [10]. Benign updates refer to normal data processing operations which may delete or change watermarked rows [10]. Malicious attacks refer to attacks where the watermark is intentionally erased [10]. False claims of ownership refer to plausible but false claims of ownership by an attacker who adds another watermark to an already watermarked database [10].

Malicious attacks can come in various forms such as bit attacks, rounding attacks, transformation attacks, subset attacks and mix-and-match attacks [10]. *Bit attacks* destroy the watermark by altering some bits [10]. *Rounding attacks* remove marks of the watermark in numeric attributes by rounding the value of attributes [10]. *Transformation attacks* transform the values of numeric attributes linearly [10]. *Subset attacks* may remove the watermark by taking a subset of attributes or tuples of watermarked data [10]. *Mix-and-match* attacks take disjoint tuples of multiple relations to create a new relation [10]. In the literature, many different malicious attacks are proposed, see [5, 11, 22, 23].

2.2.5. Fingerprinting of Relational Databases

This sub-section investigates fingerprinting (or transaction tracking) of relational databases in more detail because this is the kind of watermarking relevant for data leakage detection. There is one major difference between watermarking (for copyright protection) and fingerprinting [22]. In watermarking the same watermark is used for different data users, while in fingerprinting different watermarks are used for different data users [22]. More precisely, watermarking aims to identify a data owner, while fingerprinting aims to identify a data leaker [19]. In addition, copyright protection is attacked in a way that an attacker claims ownership or weaken a merchant’s claim, while fingerprinting is attacked in a way such that an innocent or no principal is identified as traitor [19]. In summary, watermarking and fingerprinting require different techniques due to different aims [19].

The fingerprints of an object are characteristics that may distinguish that object from similar objects [24]. With this in mind, fingerprinting is the process of adding and recording fingerprints, or the process of identifying and recording fingerprints [24]. In the application of publishing databases in a network, unauthorized duplication and distribution may be discouraged by fingerprinting each copy of the database [5]. The source of an unauthorized database copy can then be identified by extracting the embedded fingerprint [5].

Collusion attacks are unique to fingerprinting [19]. These attacks are performed by creating a new copy of the data based on multiple differently fingerprinted copies of the same data [19]. Majority and mix-and-match attacks are examples for collusion attacks. In majority attacks a new copy is created based on the majority function of different copies [23]. In mix-and-match attacks subsets of attributes and tuples from different copies are combined to create a new copy [23]. The fingerprint detection of a colluded copy may detect the fingerprint of an innocent data user or no valid fingerprint at all [19]. The collusion attack is only relevant in fingerprinting but irrelevant in watermarking [19].

We investigated existing watermarking techniques for fingerprinting of relational databases but none of them seemed promising for the application on the example medical sensor data. This insight has drawn our attention to watermarking of time series data.

2.3. Discussion and Related Work

In the course of this thesis, it turned out that neither watermarking of multimedia data nor watermarking of relational databases are directly applicable to medical sensor data. This insight is discussed below based on the differences between watermarking of multimedia and watermarking of relational data named by Agrawal et al. [10].
Medical sensor data may be similar to relational data because it consists of measurements representing separate objects and not consisting of a large number of bits with high redundancy, c.f. [10]. Medical sensor data may also be similar to multimedia because there is a temporal positioning of measurements, while in relational data there is no ordering between tuples, c.f. [10]. In the end, medical sensor data may neither be similar to multimedia nor to relational data because parts of the data can be dropped

without causing perceptual changes, and normal processing operations (insert, update, delete) are not the norm, c.f. [10]. In summary, watermarking of multimedia and watermarking of relational databases cannot be directly applied to medical sensor data.

The question arose what kind of watermarking may then be relevant for medical sensor data. Panah et al. [17] defined time series data as an ordered set of numerical values associated with timestamps. Medical sensor data as time indexed sequences of measurements may be regarded as time series data based on this definition by Panah et al. [17]. Considering medical sensor data as time series data lead our attention to watermarking of time series data.

Duy et al. [25] discussed why watermarking of multimedia cannot be adapted for biomedical time series data which confirmed our statement about less redundancy and provides additional differences. First and already mentioned, time series signals, e.g. Electroencephalography (EEG) or Electrocardiography (ECG), have less redundancy than images or audio making embedding more difficult, because there are less possibilities for hiding watermarks [25]. Second, EEG signals are fast changing time series, while audio signals have a slow time-varying property. [25]. Third, regarding biomedical time series data, the embedding of watermarks must not change the data in a way such that a patient is misdiagnosed [25]. Because of these differences, watermarking techniques of multimedia data cannot be directly used for watermarking of time series data.

After we found out that fingerprinting of relational data is not promising for medical sensor data and that medical sensor data may be regarded as time series data, we investigated existing watermarking techniques for time series data.
Existing watermarking techniques for time series data focus on, among others, tamper proofing and authentication, as shown in a review by Panah et al. [17]. With the problem scenario in mind, however, watermarking to detect data leakage, i.e. watermarking for transaction tracking (or fingerprinting), is required.
Blind watermarking techniques of biomedical time series data has been proposed by Duy et al. [25, 26]. Considering the problem scenario with medical sensor data stored in a database not being modified, we found that informed watermark embedding can be used to improve the imperceptibility of watermarks. Furthermore, informed watermark detection can be used to perform similarity searches and to extract watermarks by subtraction based on original data.
Recently, Ayday et al. [27] proposed a collusion-secure watermarking technique of sequential data (including time series data) for data leakage detection. In contrast to Ayday et al. [27], in our approach every single measurement that is shared can be part of a watermark to increase security against malicious attacks but still considering preserving the usability of the data for diagnostic purposes. In addition, small, independent and meaningful units of data (fragments) can be watermarked individually to further improve security in case of when only a single or few such units of data from a dataset are leaked.
In summary, existing watermarking techniques of time series data did not meet our expectations to overcome the problem scenario and to provide high security considering the sensitivity of medical sensor data. To this end, we introduced a new watermarking approach.

The introduced watermarking approach is mainly based on two concepts named in the text book of Cox et al. [3]. Due to medical sensor data always being available, the approach uses informed watermark embedding and informed watermark detection, see Figure 14. This means that original is analyzed before watermark generation and that original data is used to extract an embedded watermark from

watermarked data. In addition, we added a secret (watermark) key to secure the algorithm if it is publicly known based on the recommendation by Cox et al. [3].
Furthermore, we also use the watermarking properties named by Cox et al. [3] to determine requirements and design goals for the proof-of-concept prototype. In case of security, we evaluate some malicious attacks which are based on attacks to watermarks of relational databases, such as subset selection attack or rounding attacks.

In this chapter the state of the art of watermarking in general and of watermarking of relational databases was introduced. In the end, we also investigated existing watermarking techniques of time series data. In the next chapter, requirements and design goals for the proof-of-concept prototype are discussed.

3. Requirements and Design Goals

This chapter determines requirements and design goals for the proof-of-concept prototype based on the context of this thesis. The context includes the problem statement, solution idea and watermarking properties. In the end, a summary of the requirements and design goals is provided.

3.1. Requirements based on the Problem Statement

This section determines requirements for the prototype based on the problem statement to ensure that the prototype provides a solution for the problem scenario.

PS-1: The prototype shall be based on watermarking. The problem scenario requires the implementation of a watermarking approach. This requirement is fulfilled if the prototype performs watermarking according to the definition of watermarking by Cox et al. [3].

PS-2: The prototype shall be applicable to medical sensor data. The problem scenario requires an implementation of a watermarking approach of medical sensor data. This requirement is fulfilled if the prototype can watermark the example medical sensor data described in Section 1.2.

PS-3: The prototype shall detect data leakage. The problem scenario requires an implementation of a watermarking approach for data leakage detection. This requirement is fulfilled if, given a suspicious dataset, the prototype can identify whether the dataset originates from the database and, if so, who leaked the data.

3.2. Requirements based on the Solution Idea

This section determines requirements for the prototype based on the solution idea to ensure that the prototype is implemented according to the introduced watermarking approach.

SI-1: The prototype shall implement the use of dataset fragments. The watermarking approach includes storing of dataset fragments instead of datasets to enable individual watermarking of small units of data. Therefore, datasets are fragmented based on the fragmentation criteria sensor and time. In the case of diabetes data, we assumed that the fragmentation criteria are sensor and day. This requirement is fulfilled if measurements from the same sensor and from the same day are stored together in the database.

SI-2: The prototype shall implement the proposed watermark embedding. The watermarking approach includes the process of watermark embedding which consists of the generation and embedding of a watermark. This requirement is fulfilled if all its sub-requirements are fulfilled.

SI-2a: The watermark generation shall be configurable by a requested fragment. The watermark generation securely produces a matching watermark based on the secret key and the measurement values of a requested fragment. This requirement is fulfilled if watermark generation produces different watermarks for different fragments.

SI-2b: The watermark generation shall be configurable by a usability constraint. The watermark generation produces a watermark based on the error limitations of a usability constraint. The limitation aims to preserve metrics of data such as standardized metrics for continuous blood glucose monitoring by Battelino et al. [6]. This requirement is fulfilled if watermark generation produces different watermarks using different usability constraints and if watermarks are able to preserve the following metrics by Battelino et al. [6]:

- Mean Glucose [mmol/L]
- Glucose Management Indicator: 12.71 + 4.70587 x mean glucose [mmol/L] [28]
- Glycemic Variability: mean glucose ± standard deviation [mmol/L] [29]
- Time above Range: > 13.9 [mmol/L]
- Time above Range: 10.1 – 13.9 [mmol/L]
- Time in Range: 3.9 – 10.0 [mmol/L]
- Time below Range: 3.0 – 3.8 [mmol/L]
- Time below Range: < 3.0 [mmol/L]

SI-2c: The watermark generation shall be configurable by a watermark number. The watermark generation produces a watermark based on a watermark number which depends on previous requests stored in the request log. This requirement is fulfilled if watermark generation produces different watermarks using different watermark numbers.

SI-2d: The watermark generation shall generate a matching watermark. The watermark generation produces a matching watermark as a sequence of matching errors with the same length as the requested fragment. This requirement is fulfilled if the result of watermark generation is a matching watermark.

SI-2e: The generated watermark shall be embedded. The generated watermark is embedded in the requested fragment by adding it to the measurement values which results in a watermarked fragment. This requirement is fulfilled if the result of embedding is a watermarked fragment.

SI-3: The prototype shall implement the proposed watermark detection. The watermarking approach includes the process of watermark detection which consists of fragment similarity search, watermark extraction, watermark re-generation, watermark similarity search and leaker identification. This requirement is fulfilled if all its sub-requirements are fulfilled.

SI-3a: The fragment similarity search shall identify a matching fragment. The fragment similarity search identifies the matching fragment of a suspicious fragment by matching the suspicious fragment against a set of relevant original fragments. This requirement is fulfilled if the result of fragment similarity search is a matching fragment.

SI-3b: The embedded watermark shall be extracted. The watermark extraction extracts an embedded watermark from a suspicious fragment by subtracting the measurement values of the matching fragment. This requirement is fulfilled if the result of extraction is a watermark.

SI-3c: The already embedded watermarks shall be re-generated. The already embedded watermarks of a matching fragment are re-generated for watermark similarity search. This requirement is fulfilled if the result of watermark re-generation is a set of already embedded watermarks.

SI-3d: The watermark similarity search shall identify matching watermarks. The watermark similarity search identifies matching watermarks of an extracted watermark by matching the extracted watermark against a set of re-generated watermarks. This requirement is fulfilled if the result of watermark similarity search are matching watermarks.

SI-3e: The leaker identification shall identify potential data leakers. The leaker identification identifies potential data leakers of a suspicious datasets by combining the matching watermarks associated to the same data users. This requirement is fulfilled if the result of leaker identification is a list of potential data leakers together with their leakage probabilities.

3.3. Requirements and Design Goals based on Watermarking Properties

This section determines requirements and design goals for the prototype based on watermarking properties named by Cox et al. [3], see Section 2.1.2.

WP-1: The embedding effectiveness shall be 100 %. The embedding effectiveness, which is the probability to embed a watermark in a random fragment [3], is required to be 100 % because each fragment needs to be watermarked individually. This requirement is fulfilled if the probability of successfully embedding a watermark into a randomly selected fragment is 100 %.

WP-2: The fidelity should be preferably high. The fidelity, which is the similarity between an original and its watermarked fragments [3], is required to be preferably high because a domain expert may identify and remove conspicuous parts of a watermark. Furthermore, if the data is used for diagnostic purposes it could be misinterpreted due to introduced distortions by watermarking [25]. The fidelity may depend on the usability constraint which limits the errors of a watermark. The achievement of this design goal is measured by the similarities between an original fragment and its watermarked fragments with the similarities being computed by fragment similarity search of watermark detection.

WP-3: The data payload should be as high as necessary. The data payload, which is the amount of information encoded in a watermark [3], is only required to be as high as necessary such that a watermark is associated to a single data user. The payload may depend on the watermark number which is encoded as a watermark. The achievement of this design goal is measured by the number of bits encoded in a watermark.

WP-4: The detection shall be informed. Informed detection, which is the usage of information about the original fragment during watermark detection [3], is required for fragment similarity search and watermark extraction. Due to data stored in the database not being modified anymore, informed detection is possible. This requirement is fulfilled if original fragments are utilized during watermark detection.

WP-5: The false positive rate should be preferably low. The false positive rate, which is the frequency of falsely detected watermarks in unwatermarked fragments [3], is required to be preferably low to avoid false theft accusations.
There are two kinds of unwatermarked objects, namely, an unwatermarked dataset and an unwatermarked fragment. In case of an unwatermarked fragment, a false positive detection can occur, if the unwatermarked fragment is very similar to a stored fragment and one of its watermarks. Nevertheless, the detection of a watermark in a single fragment does not reliably identify a data leaker. In case of an unwatermarked dataset, it may be very unlikely that all unwatermarked fragments of a dataset are very similar to stored fragments and the watermark combination of a data user.
In this thesis, the false positive rate in single unwatermarked fragments is investigated. False positives may depend on the number of relevant fragments in the database and the number of watermarks for each fragment. The achievement of this design goal is measured by the frequency of detecting a matching watermark in unwatermarked fragments.

WP-6: The robustness should be preferably high. The robustness, which is the ability of a watermark to withstand normal processing operations [3] is required to be preferably high, because this is the basis for the watermark to be secure against malicious attacks. The achievement of this design goal is measured together with the closely related WP-7.

WP-7: The security should be preferably high. The security, which is the ability of a watermark to withstand malicious attacks [3], is required to be preferably high, because dealing with sensitive patient data requires high privacy. There are different kinds of malicious attacks that can be used to destroy a watermark. The coverage of all attacks or possible attack combinations would go beyond the scope of this thesis. Therefore, five interesting attacking methods for this application are defined in the sub-design goals. The achievement of this design goal is measured by its sub-design goals.

WP-7a: The prototype should be secure against rounding value attacks. The rounding value attack rounds values of measurements based on a decimal digit. On the one hand, the watermark may be lost due to rounded values, on the other hand the attack can make the data unusable by rounding at a too high extend. The achievement of this sub-design goal is measured by watermark similarities of a watermarked fragment which was target of rounding value attacks using different configurations.

WP-7b: The prototype should be secure against random value attacks. The random value attack changes values of measurements randomly based on a maximum error. On the one hand, the watermark may be damaged by randomly changed values, on the other hand the attack may eventually make the data unusable if the random changes are too high. The achievement of this sub-design goal is measured by watermark similarities of a watermarked fragment which was target of random value attacks using different configurations.

WP-7c: The prototype should be secure against deletion attacks. The deletion attack removes measurements of a dataset based on a frequency. The deletion of measurements removes parts of an embedded watermark. The achievement of this sub-design goal is measured by watermark similarities of a watermarked fragment which was target of deletion attacks using different configurations.

WP-7d: The prototype should be secure against subset selection attacks. The subset selection attack selects a subset of measurements from a fragment based on a start and an end index. The selection of a measurement subset removes parts of the embedded watermark. The achievement of this sub-design goal is measured by watermark similarities of a watermarked fragment which was target of subset selection attacks using different configurations.

WP-7e: The prototype should be secure against mean collusion attacks. The mean collusion creates a new dataset by computing the mean of values of corresponding measurements from differently watermarked datasets. The computation of the mean of values hides the watermark of each colluder. The achievement of this sub-design goal is measured by watermark similarities of a colluded fragment which was target of mean collusion attacks using different configurations.

WP-8: Cipher keys shall not be used, but watermark keys shall be used. Cipher keys to control watermark encryption [3] are not required because no information is encrypted in the watermark. Watermark keys to control watermark embedding and detection are required to ensure security of the algorithm if it is publicly known [3]. This requirement is fulfilled if no encryption is performed and if watermarking is secured by secret watermark keys.

WP-9: Modification and multiple watermarks shall not be used. The modification, which is the modifiability of already embedded watermarks [3], and multiple watermarks, which is the embedding of multiple watermarks in a single fragment [3], are not required because of the following reasons. First, the only purpose of watermarking is to identify a data user, which requires a single watermark that is embedded only once. Second, modifications or multiple watermarks distort the initial watermark so that a data user cannot be identified. This requirement is fulfilled if watermarks are not modified and if fragments are watermarked only once per request.

WP-10: The prototype should be preferably cost-efficient. The costs, which is the computation cost of watermark embedding and detection [3], are required to be preferably low because performance can be an important aspect for practical usage. In terms of cost-efficiency, two issues are of interest, namely, the performance of embedding and detection, and the required number of embedders and detectors [3]. In this thesis, only the embedding and detection performance are investigated. In addition, fragmentation performance when storing data is also investigated, because fragmentation is additional effort specific to the implemented watermarking approach. This design goal is measured by its sub-design goals.

WP-10a: The fragmentation should be preferably fast. The fragmentation is required to be preferably fast because it is additional effort specific to the implemented watermarking approach. The fragmentation performance may depend on the number of fragments of a to-be stored dataset. This sub-design goal is measured by the time required for fragmentation of differently sized datasets.

WP-10b: The watermark embedding should be preferably fast. The watermark embedding is required to be preferably fast because data users requesting data may want to have fast access to the requested data.
There is a distinction between two use cases; a data user like a doctor who requests data of a patient of a single day, week or a month and a data user like a researcher who requests data of many patients over a longer period of time. In the first use case, the embedding should be performed within a few seconds, because a typical doctor may want to access the data immediately while talking to a patient. In the second use case, the embedding should be performed within a few minutes or hours, because a typical researcher may not have a strict time constraint like a doctor. Nevertheless, both use cases require fast embedding.
The embedding performance may depend on the number of requested fragments. This sub-design goal is measured by the time required for watermark embedding of differently sized datasets.

WP-10c: The watermark detection should be as fast as necessary. The watermark detection is only required to be as fast as necessary because it is not used that frequently and no one is dependent on it to keep working. Furthermore, no disadvantage arises due to a slow detection time – the detection may even become more accurate.
There are two kinds of detection performances, namely fragment detection performance and dataset detection performance. The fragment detection performance may depend on the number of relevant fragments in the database and the number of already embedded watermarks of the matching fragment. The dataset detection performance is the combined fragment detection performance of all fragments of the dataset. This sub-design goal is measured by the time required for watermark detection of a suspicious fragment using different database contents.

In this chapter, the requirements and design goals for the proof-of-concept prototype were determined. Table 4 summarizes these requirements and design goals and, in addition, provides their evaluation criteria. In the next chapter, the design of watermark embedding and detection is presented.

Requirements based on the Problem Statement		
PS-1	The prototype shall be based on watermarking.	Decision [Y/N]
PS-2	The prototype shall be applicable to medical sensor data.	Decision [Y/N]
PS-3	The prototype shall detect data leakage.	Decision [Y/N]
Requirements based on the Solution Idea		
SI-1	The prototype shall implement the use of dataset fragments.	Decision [Y/N]
SI-2	The prototype shall implement the proposed watermark embedding.	
SI-2a	The watermark generation shall be configurable by a requested fragment.	Decision [Y/N]
SI-2b	The watermark generation shall be configurable by a usability constraint.	Decision [Y/N]
SI-2c	The watermark generation shall be configurable by a watermark number.	Decision [Y/N]
SI-2d	The watermark generation shall generate a matching watermark.	Decision [Y/N]
SI-2e	The generated watermark shall be embedded.	Decision [Y/N]
SI-3	The prototype shall implement the proposed watermark detection.	
SI-3a	The fragment similarity search shall identify a matching fragment.	Decision [Y/N]
SI-3b	The embedded watermark shall be extracted.	Decision [Y/N]
SI-3c	The already embedded watermarks shall be re-generated.	Decision [Y/N]
SI-3d	The watermark similarity search shall identify matching watermarks.	Decision [Y/N]
SI-3e	The leaker identification shall identify potential data leakers.	Decision [Y/N]
Requirements and Design Goals based on Watermarking Properties		
WP-1	The embedding effectiveness shall be 100%.	Decision [Y/N]
WP-2	The fidelity should be preferably high.	Similarity [%]
WP-3	The data payload should be as high as necessary.	Size [Bits]
WP-4	The detection shall be informed.	Decision [Y/N]
WP-5	The false positive rate should be preferably low.	Frequency [%]
WP-6	The robustness should be preferably high.	WP-7
WP-7	The security should be preferably high.	
WP-7a	The prototype should be secure against rounding value attacks.	Similarity [%]
WP-7b	The prototype should be secure against random value attacks.	Similarity [%]
WP-7c	The prototype should be secure against deletion attacks.	Similarity [%]
WP-7d	The prototype should be secure against subset selection attacks.	Similarity [%]
WP-7e	The prototype should be secure against mean collusion attacks.	Similarity [%]
WP-8	Cipher keys shall not be used, but watermark keys shall be used.	Decision [Y/N]
WP-9	Modification and multiple watermarks shall not be used.	Decision [Y/N]
WP-10	The prototype should be preferably cost-efficient.	
WP-10a	The fragmentation should be preferably fast.	Time [Seconds]
WP-10b	The watermark embedding should be preferably fast.	Time [Seconds]
WP-10c	The watermark detection should be as fast as necessary.	Time [Seconds]

Table 4: Summary of Requirements and Design Goals

4. Design

This chapter describes the detailed design of watermarking embedding and detection for the proof-of-concept prototype. Therefore, the tasks of watermark embedding and detection of the introduced watermarking approach are instantiated with concrete algorithms.

4.1. Watermark Embedding

Whenever an authorized data user requests data, watermark embedding is performed which consists of the tasks watermark generation and embedding. Before watermark generation, the requested fragments corresponding to the requested data are retrieved from the database.

Watermark Generation. The watermark generation produces a matching watermark for a requested fragment. For each requested fragment, Steps 1-3 are performed, and for each measurement of a requested fragment, Steps 4-6 are performed.

Step 1: Retrieving Parameters. In order to perform the watermark generation, the requested fragment, its corresponding usability constraint and the current watermark number are required. The requested fragment is already available. The usability constraint is retrieved from the database based on the type and unit of measurements of the requested fragment. The watermark number is determined by looking up the request log. If the data user has not requested this fragment yet, the next watermark number is computed based on the highest already used watermark number for this fragment. If the data user has already requested this fragment, the same watermark number is used.

Step 2: Configuring the Error Selection. After the required parameters are retrieved, watermark generation is configured by the secret key of the requested fragment and the watermark number. First, a pseudo random number generator (PRNG) is seeded by the secret key of the requested fragment. Then, the PRNG is re-seeded by the concatenation of the watermark number and a random integer which is generated by the PRNG. Because the PRNG is later used to select errors, watermark generation is secured, and the watermark to-be generated is determined.

Step 3: Computing the Error Distribution. The usability constraint's number of ranges configures the error distribution by indicating the number of error sub-ranges within the possible error range. Because the selection probabilities of error sub-ranges are equal for each measurement of the requested fragment and independent of their size, they are computed only once in the beginning. The selection probability of an error sub-range *i* is computed based on Equation 1.

$$for\ i = 0 \dots \left(\frac{number\ of\ ranges}{2}\right) - 1$$

$$if\ i = 0\ then\ P_i = \frac{0{,}5}{2^{\left(\frac{number\ of\ ranges}{2}\right)-(i+1)}}\ else\ P_i = \frac{0{,}5}{2^{\left(\frac{number\ of\ ranges}{2}\right)-i}}$$

Equation 1: Computation of Sub-Ranges Selection Probabilities

Let us discuss the first line of Equation 1. The possible error range is later divided into a lower and an upper error range with, in turn, each of them being divided into half of the error sub-ranges. To this end, the number of ranges must be dividable by two. Each half of the error sub-ranges has a total selection probability of 50 % together achieving 100 %. Because the selection probabilities of error-sub ranges in the lower and upper error range are identical, only the selection probabilities of one half of the error sub-ranges are computed.

Now, let us discuss the second line of Equation 1. In the first case (*i* = 0*)* the selection probability of the last error sub-range which is identical to the second last error sub-range is computed to achieve a total of 50 %. In the other cases (*i* > 0) the selection probabilities of all other error sub-ranges are computed. Using this equation, the probability of an error sub-range to be selected decreases based on its distance to the original value except with a number of ranges of 4 in which all error sub-ranges have a selection probability of 25 %. In addition, the example selection probabilities of error sub-ranges shown in Table 1 from the solution idea can be computed using Equation 1.

Step 4: Computing the Error Range. After computing the selection probabilities of the error sub-ranges, the error range of each measurement is computed. The lower bound of the initial error range is computed by subtracting the maximum error from the original measurement value. The upper bound of the initial error range is computed by adding the maximum error to the original measurement value. If the initial lower bound is below the sensor's minimum value, the sensor's minimum value is the new lower bound. If the initial upper bound exceeds the sensor's maximum value, the sensor's maximum value is the new upper bound.
Now, the value structure of the previously watermarked measurement value $v_w(t\text{-}1)$, the to-be watermarked measurement value *v(t)* and the next measurement value *v(t+1)* is considered. Table 5 provides the possible value structures together with preserving actions. The preserving actions may modify the lower and upper bound in a way that the final error range preserves the value structure. There are two exceptions in which preserving a part of the value structure is not desired, namely, if $v_w(t\text{-}1\ ) = v(t)$ and *if v(t) = v(t+1)*. The final error range consists of all values between the lower and upper bound.

Case	Value Structure	Preserving Actions
1	$v_w(t-1) < v(t) < v(t+1)$	$if\ v(t)_{lowerBound} < v_w(t-1)\ then\ v(t)_{lowerBound} = v_w(t-1)$ $if\ v(t)_{upperBound} > v(t+1)\ then\ v(t)_{upperBound} = v(t+1)$
2	$v_w(t-1) > v(t) > v(t+1)$	$if\ v(t)_{lowerBound} < v(t+1)\ then\ v(t)_{lowerBound} = v(t+1)$ $if\ v(t)_{upperBound} > v_w(t-1)\ then\ v(t)_{upperBound} = v_w(t-1)$
3	$v_w(t-1) < v(t) > v(t+1)$	$if\ v(t)_{lowerBound} < v_w(t-1)\ then\ v(t)_{lowerBound} = v_w(t-1)$ $if\ v(t)_{lowerBound} < v(t+1)\ then\ v(t)_{lowerBound} = v(t+1)$
4	$v_w(t-1) > v(t) < v(t+1)$	$if\ v(t)_{upperBound} > v_w(t-1)\ then\ v(t)_{upperBound} = v_w(t-1)$ $if\ v(t)_{upperBound} > v(t+1)\ then\ v(t)_{upperBound} = v(t+1)$
5	$v_w(t-1) = v(t) < v(t+1)$	$if\ v(t)_{upperBound} > v(t+1)\ then\ v(t)_{upperBound} = v(t+1)$
6	$v_w(t-1) = v(t) > v(t+1)$	$if\ v(t)_{lowerBound} < v(t+1)\ then\ v(t)_{lowerBound} = v(t+1)$
7	$v_w(t-1) < v(t) = v(t+1)$	$if\ v(t)_{lowerBound} < v_w(t-1)\ then\ v(t)_{lowerBound} = v_w(t-1)$
8	$v_w(t-1) > v(t) = v(t+1)$	$if\ v(t)_{upperBound} > v_w(t-1)\ then\ v(t)_{upperBound} = v_w(t-1)$
9	$v_w(t-1) = v(t) = v(t+1)$	No Actions.

Table 5: Possible Value Structures and their preserving Actions

Step 5: Computing the Error Sub-Ranges. The error sub-ranges of a measurement are computed based on the final error range which was computed in Step 4. The error range below the original value is the lower error range and the error range above the original value is the upper error range. The error sub-ranges indicated by the number of ranges are halved, with one half being distributed to the lower error range and the other half to the upper error range. The lower error sub-ranges may be differently sized

than the upper error sub-ranges due to the lower error range possibly being differently sized than the upper error range.
The selection probabilities computed in Step 3 are assigned to the corresponding error sub-ranges. In case of the lower or upper error range being zero, its error sub-ranges are not considered for selecting an error to drastically decrease the probability of an error to be 0. Error sub-ranges are not considered for error selection if their selection probabilities are 0 % and if the selection probabilities of the other error sub-range are doubled to achieve 100 %. Table 2 from the solution idea provided an example of error sub-ranges together with their selection probabilities.

Step 6: Selecting the Error. After the error sub-ranges of a measurement are computed, finally, the error is selected by using the PRNG from Step 2. The PRNG pseudo randomly selects an error sub-range based on the selection probabilities and within the selected error sub-range, the PRNG pseudo randomly selects an error. The selected error is added to the watermark's sequence of errors. If the requested fragment consists of another measurement, Steps 4-6 are repeated.

Embedding. The generated watermark is embedded by adding the watermark to the requested fragment. The following steps are performed for each requested fragment.

Step 1: Retrieving Parameters. In order to perform embedding, the requested fragment and the generated watermark are required. The requested fragment is already available and, after watermark generation, also the generated watermark is available.

Step 2: Adding the Errors. The watermark is embedded by adding each error of the watermark to the corresponding measurement value of the requested fragment, as shown in Equation 2. The result of embedding is a watermarked fragment with the watermark being associated to the requesting data user.

$$for\ i = 1 \ldots number\ of\ measurements$$
$$watermarked\ measurement\ value_i = measurement\ value_i + error_i$$

Equation 2: Embedding of Watermarks

In the end, the requested watermarked fragments are combined and transmitted to the requesting data user.

4.2. Watermark Detection

Whenever a suspicious dataset is found, watermark detection can be performed which consists of the tasks fragment similarity search, extraction, watermark re-generation, watermark similarity search and leaker identification. Before watermark fragment similarity search, the suspicious dataset is fragmented based on the fragmentation criteria which, in the case of continuous blood glucose, is sensor and day resulting in a set of suspicious fragments.

Fragment Similarity Search. The fragment similarity search identifies the matching original fragment of a suspicious fragment. For each suspicious fragment, Step 1 and Step 4 are performed, and for each original fragment from the set of relevant fragments, Steps 2-3 are performed.

Step 1: Retrieving Parameters. In order to perform fragment similarity search, the suspicious fragment, its corresponding usability constraint and a set of relevant original fragments are required. The suspicious fragment is already available. The usability constraint is retrieved from the database based on the type and unit of measurements of the suspicious fragment. The set of relevant original fragments is retrieved from the database by requesting all fragments with measurements of the same type and unit as the suspicious fragment.

Step 2: Matching Measurements Analysis. The matching measurement analysis identifies matching measurements between the suspicious fragment and a relevant original fragment. Every measurement of the suspicious fragment is compared with every measurement of an original fragment. A matching measurement is identified, if the timestamp of a suspicious measurement is equal to the timestamp of an original measurement. The result of matching measurements analysis is a list of indices referring to the matching measurements between the suspicious fragment and an original fragment.

Step 3: Computing Fragment Similarity. After the matching measurements between the suspicious fragment and an original fragment are identified, the fragment similarity is computed. For each matching measurement, the distance is computed by subtracting the value of the original measurement from the value of the suspicious measurement. The absolute distance is then relativized by the sensor's maximum value. The subtraction of the relative distance from the relative value 1 results in the measurement similarity. The fragment similarity is the summarized measurement similarity divided by the number of suspicious measurements such that not matching measurements also impact the fragment similarity. Equation 3 provides the formula for computing the fragment similarity.

$$Similarity_{Fragment} = \frac{\sum_{i=1}^{matching\ measurements} 1 - \frac{|suspicious\ value_i - original\ value_i|}{sensor's\ maximum\ value}}{number\ of\ suspicious\ measurements}$$

Equation 3: Computation of Fragment Similarity

Step 4: Determining the Matching Fragment. After performing Step 2 and 3 for each relevant original fragment, the matching fragment is determined. The original fragment with the highest similarity is selected and if it is above a user-defined threshold, the matching fragment is identified. If no fragment exceeds the user-defined similarity threshold, the suspicious fragment might not originate from the database and the watermark detection of this suspicious fragment is cancelled.

Extraction. The embedded watermark is extracted by subtracting the matching fragment from the suspicious fragment. The following steps are performed to extract the watermark.

Step 1: Retrieving Parameters. In order to extract the watermark, the suspicious fragment and the matching fragment are required. The suspicious fragment is already available. The matching fragment is available if fragment similarity search identified a matching fragment.

Step 2: Subtracting the Matching Fragment. The watermark is extracted by subtracting the measurement values of the matching fragment from their matching measurement values of the suspicious fragment, as shown in Equation 4. The result of the extraction is an extracted and eventually noisy watermark which eventually identifies a data user.

$$for\ i = 1 \ldots matching\ measurements$$
$$error_i = suspicious\ value_i - original\ value_i$$

Equation 4: Extraction of Watermarks

Watermark Re-Generation. The already embedded watermarks of a matching fragment are re-generated based on previous requests. The following steps are performed to re-generate the set of watermarks.

Step 1: Retrieving Parameters. In order to re-generate the already embedded watermarks, the matching fragment, its corresponding usability constraint and previous requests are required. The matching fragment is available if the fragment similarity search identified a matching fragment. The usability constraint has already been retrieved from the database. The previous requests of the matching fragment are retrieved from the request log. The previous requests include the already embedded watermark numbers and their associated data users.

Step 2: Generating the Set of Watermarks. The already embedded watermarks are re-generated by watermark generation identical to watermark generation of watermark embedding. For each existing request on the matching fragment, the corresponding watermark is re-generated and, together with its associated data user, collected in a set of watermarks.

Step 3: Enriching the Set of Watermarks (Optional). The set of re-generated watermarks can be enriched by computing watermark combinations. The user-defined number of colluders indicates the maximum number of watermarks that are included in watermark combinations. We only consider mean watermark combinations that are computed by the mean of errors of multiple re-generated watermarks. The resulting watermark combinations are added to the set of watermarks.

Watermark Similarity Search. The watermark similarity search identifies matching watermarks of the extracted watermark. For each extracted watermark, Step 1 is performed, and for each watermark from the set of re-generated watermarks, Steps 2-3 are performed.

Step 1: Retrieving Parameters. In order to perform watermark similarity search, the extracted watermark, the usability constraint and the set of watermarks are required. The extracted watermark is available if watermark extraction was performed. The usability constraint has already been retrieved from the database. The set of watermarks is available if watermark re-generation was performed.

Step 2: Computing Watermark Similarity. The watermark similarity is computed for each watermark from the set of re-generated watermarks based on the matching measurements. For each matching measurement, the distance is computed by subtracting the original error from the extracted error. The absolute distance is then relativized by the maximum distance (maximum error multiplied by 2). The subtraction of the relative distance from the relative value 1 results in the error similarity. The watermark similarity is the summarized error similarities divided by the number of matching measurements. In contrast to the formula for fragment similarity, non-matching measurements are not considered, because they are not part of the extracted watermark. Equation 5 provides the general formula for watermark similarity.
There is an exception to this formula, namely if an error similarity is below 0 which is the case if the extracted error is very high, i.e. ≥ three times maximum error. If this is the case, this error similarity is set from below 0 to 0 such that an outlier does not decrease the final watermark similarity at a too high extent.

$$Similarity_{Watermark} = \frac{\sum_{i=1}^{matching\ measurements} 1 - \frac{|extracted\ error_i - original\ error_i|}{2 * maximum\ error}}{number\ of\ matching\ measurements}$$

Equation 5: Computation of Watermark Similarity

Step 3: Determining Matching Watermarks. The watermark matches the extracted watermark if its watermark similarity is above a user-defined threshold. If the watermark is identified as a matching watermark, the associated data user is a potential leaker of the suspicious fragment and possibly of the suspicious dataset. In this case, the matching watermark is stored in a set of matching watermarks. If the watermark is not identified as a matching watermark, the associated data user is not considered for data leakage of the suspicious fragment anymore.

Leaker Identification. The leaker identification identifies potential data leakers of the suspicious dataset based on the set of matching watermarks from watermark similarity searches. The following steps are performed to identify potential data leakers.

Step 1: Retrieving Parameters. In order to perform leaker identification, the number of suspicious fragments and the set of matching watermarks are required. The number of suspicious fragments is already available. The set of matching watermarks is available if watermark similarity search has been performed for each fragment of the suspicious dataset.

Step 2: Computing Leakage. The leakage probability of a data user is computed by its averaged watermark similarities, as shown in Equation 6. The watermark similarities associated with the same data user are summarized and divided by the number of suspicious fragments. The resulting leakage probabilities are used to rank the potential data leakers of the suspicious dataset.

$$P_{Leaker} = \frac{\sum_{i=1}^{number\ of\ suspicious\ fragments} Similarity_{Watermark_i}}{number\ of\ suspicious\ fragments}$$

Equation 6: Computation of Leakage Probability

In the end, the ranking of potential data leakers of each suspicious fragment and of the suspicious dataset are provided in a detection report.

4.3. Expected Results and Limitations

This section discusses expected results and limitations regarding the implementation and evaluation of the proof-of-concept prototype.

Watermark Embedding. The following issues are related to watermark embedding, namely, optimizing watermark generation, performance of watermark generation and the use of watermark keys.

Optimizing Watermark Generation. Optimized watermark generation may generate very well matching watermarks ensuring high imperceptibility and maximally different watermarks simplifying watermark detection. In this thesis, watermark generation only produces matching and different watermarks because optimizing watermark generation is beyond the scope of this thesis. The imperceptibility of the watermarks may therefore not be ideal, and the detection may suffer from eventually similar watermarks. In order to provide maximally different watermarks, the number of different watermarks for each fragment must be limited.
The maximum number of watermarks, for example, is 256 for each fragment. In the worst case, a fragment is requested by more than 256 data users which results in a multiple usage of watermarks. Hence, a data user is not reliably identifiable. Considering, a requested dataset typically consists of many more fragments, a data user is reliably identifiable by combining multiple watermarks. To continue this example, the combination of two fragments, each with 256 generated watermarks, results in 65 536 (256^2) unique watermark combinations, the combination of four fragments results in 4 294 967 296 (256^4) unique watermark combinations, and so on. This is sufficient to reliably identify a data user.

Performance of Watermark Generation. The watermark generation produces watermarks of requested fragments just in time. The bottleneck of watermark embedding therefore can be watermark generation. In order to solve this issue, a few watermarks for each fragment may already be pre-generated immediately after a patient uploaded its dataset. In addition, the watermarking system may maintain a small set of pre-generated watermarks. Another idea would be that the error ranges of measurements are pre-computed and stored together with the fragments. If a fragment is then requested, watermark generation only has to select the errors from the pre-computed error ranges.

Watermark Key. The watermark keys of requested fragments secure watermark generation and are stored together with the fragments. Instead of storing the secret watermark keys in the database, another strategy for management of secure keys can be employed. The secret key of a fragment, for example, can be generated just in time based on a secret base key and metadata of the fragment.

Watermark Detection. The following issues are related to watermark detection, namely, selection of relevant fragments, optimizing watermark detection, matching fragments, set of watermarks and similarity formulas.

Selection of Relevant Fragments for Fragment Similarity Search. Every fragment in the database with measurements of the same type and unit as the measurements of the suspicious fragment is considered a relevant fragment. This can improve the result of detection but decreases performance of detection for every additional fragment with measurements of this type and unit in the database. In order to solve this issue, criteria for fragments to be relevant may be applied. Such a criteria can include statistical measures such as mean values but must consider that mean values may be modified due to malicious attacks. Another idea is that for a few suspicious but representative fragments of a suspicious dataset all fragments in the database are relevant fragments. Furthermore, for all other suspicious fragments of the dataset the set of relevant fragments is reduced based on the matching fragments of the suspicious representative fragments.

Optimizing Watermark Detection. The fragment similarity search, the watermark extraction and the watermark similarity search are based on the result of matching measurements analysis, which relies on comparing the timestamps of measurements. Malicious attacks on the time dimension, e.g. time shifting attacks, are therefore always successful. Beside that vulnerability, the time dependent detection may be faster, because it simply relies on comparisons of timestamps. This may also improve detection, because no matching measurements are identified between fragments consisting of measurements with different timestamps. Ideally, watermark detection is optimized by using a matching measurements analysis which is not only time dependent or which is even time independent. In addition, watermark detection can be optimized by improving security against known attacking methods or attack combinations.

Matching Fragments. The fragment similarity search identifies the matching fragment of a suspicious fragment if it has the highest fragment similarity and its similarity exceeds a user-defined threshold. For the sake of simplicity, only a single matching fragment is considered. In practice, multiple fragments with high similarity can be considered as matching fragments. In case of two or more fragments being very similar, this may improve detection, but it may also decrease detection performance tremendously.

Set of Watermarks. The set of watermarks used in the watermark similarity search consists of already embedded watermarks of the matching fragment. In order to improve detection against mean collusion attacks, the prototype enriches the set of watermarks by adding mean watermark combinations. In addition, watermark detection may enrich the set of watermarks with other watermark combinations or extend existing watermarks with different versions of the watermark, e.g. watermarks after rounding value attacks using different configurations. Another idea is to start combining watermarks only if watermark similarity search could not identify matching watermarks. The disadvantage of computing watermark combinations is the exponential computation effort for every additional watermark of the matching fragment.

Similarity Formulas. The formulas for fragment and watermark similarity are based on the distance between matching measurements. In case of time series data, there may be more accurate ways to determine the similarity between two sequences.

Evaluation. The following issues are related to the evaluation of the requirements and design goals, namely, the attacking methods and the test data.

Attacking Methods. The watermark detection is only evaluated regarding five interesting attacking methods, because the evaluation of further attacking scenarios goes beyond the scope of this thesis. In addition, the detection is not designed to be secure against all kinds of attacks, as already mentioned. In practice, the prototype could be successfully attacked by other attacking methods or combinations of attacks. Time shifting attacks, for example, are always successful using a time dependent detection.

Test Data. The only data available for evaluation is the example medical sensor data described in Section 1.2. The example data consists of 59 424 measurements or, in other words, 224 fragments of different days. This data is sufficient for functional evaluation, but for detection performance evaluation, additional fragments are required. Besides that, the evaluation may in some cases depend on the test data and the content of the database, because of using similarity searches for watermark detection.

In this chapter, the detailed design of watermark embedding and detection was presented, and expected results as well as limitations were discussed. The proof-of-concept prototype is implemented based on the algorithms proposed in this chapter.

5. Implementation

This chapter briefly presents the implementation of the proof-of-concept prototype. The platform of the prototype and its main levels are introduced, namely, the database, services, simulators and interfaces.

5.1. Platform

This section presents the platform of the implemented proof-of-concept prototype. The platform is based on the problem scenario and the solution idea of this thesis. The prototype is implemented as a Java console application and utilizes the file system for data exchange to reduce any additional complexity which is not necessary for evaluation. It can be executed to perform various operations using different configurations. The source code of the prototype is available in Appendix B Source Code.

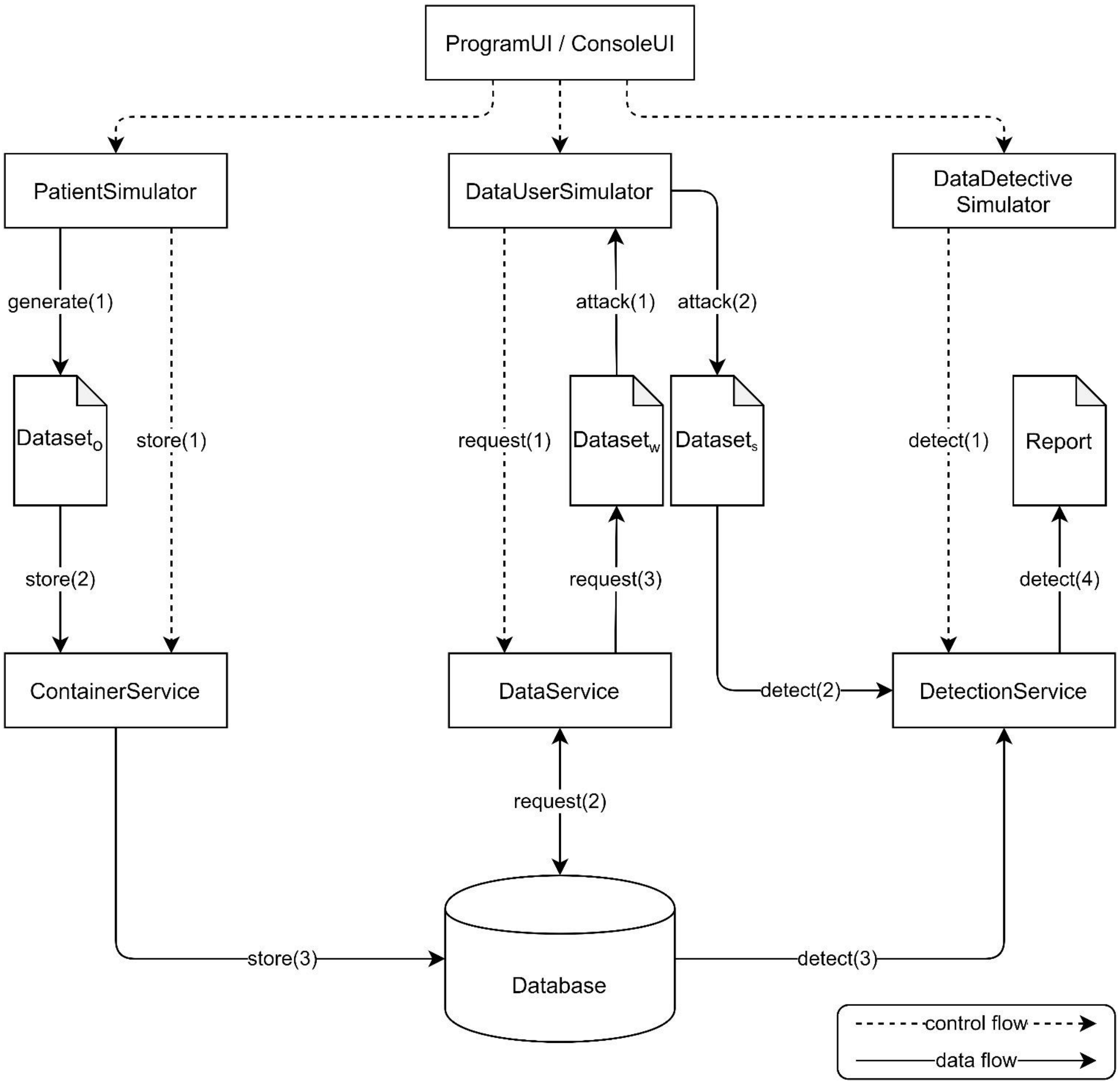


Figure 15: Illustration of Data and Control Flow in the Proof-of-Concept Prototype Platform

The platform of the prototype, which is illustrated in Figure 15, basically consists of 4 levels, from the bottom to the top, the database, services, simulators and user interface. The database stores usability constraints, requests and fragments.
The services can access the database for storing and retrieving data. The container service fragments and stores datasets. The data service retrieves requested fragments from the database, performs watermark embedding and returns the watermarked fragments. The detection service retrieves fragments from the database and performs watermark detection on suspicious datasets.
The simulators represent the roles using the services based on the problem scenario and solution idea. The patient simulator represents a patient; it provides a method for generating datasets and a method for storing datasets by using the container service. The data user simulator represents a data user; it provides a method for requesting datasets by using the data service and it provides methods for attacking (watermarked) datasets. The data detective simulator represents someone discovering suspicious datasets; it provides a method for detecting data leakers of suspicious datasets by using the detection service.
The simulators are configured via user interfaces. The program user interface is used when executing the java program in a development environment, while the console user interface is used when executing the jar file.

5.2. Database

The database is realized using PostgreSQL[3] which is an open source relational database management system. The database consists of three tables, namely, the usability constraint, fragment, and request table, as shown in Figure 16. For the sake of simplicity, we refrain from using foreign keys and the use of more tables, e.g. a table for data users. The SQL script to create these tables is available in Appendix A Installation Guide.

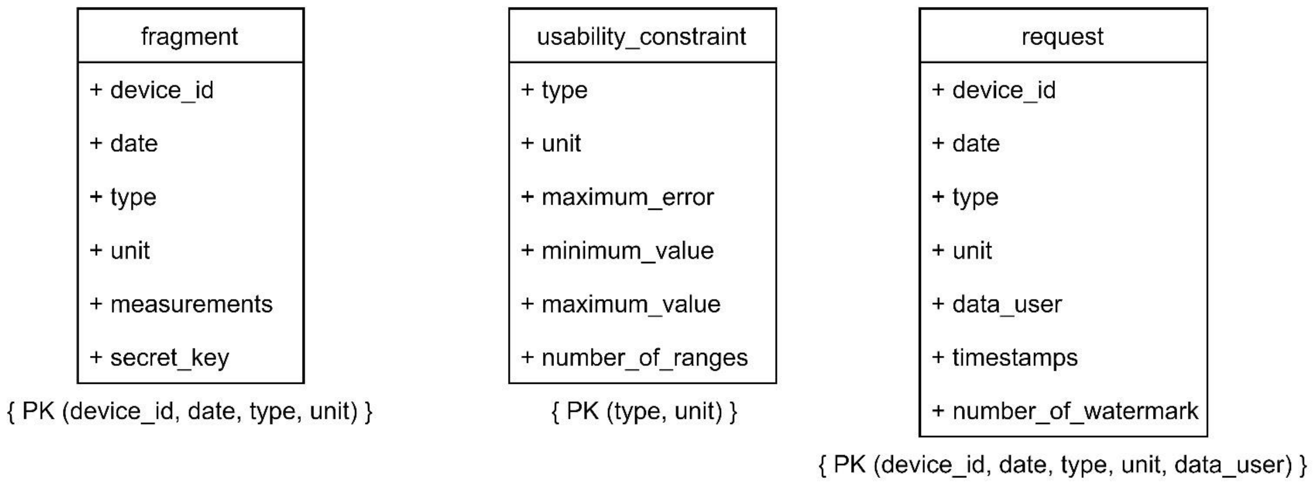


Figure 16: Tables of the Database

Fragment. The fragment table stores dataset fragments of a dataset. A fragment is identified by the unique combination of the device identifier, date, type and unit, realizing the concept of dataset fragments from the solution idea with fragmentation criteria sensor and day. Furthermore, with device identifier, date, type, unit and measurements, the fragment table realizes the conceptual schema of medical sensor data from Figure 1. In addition, the fragment table stores the secret (watermark) keys of fragments. The measurements of fragments are actually stored as raw JSON in the measurements column.

[3] PostgreSQL: https://www.postgresql.org/

Usability Constraint. The usability constraint table stores the usability constraint of each unique combination of type and unit, realizing the conceptual schema of usability constraints from Figure 9. A usability constraint consists of the maximum error, the sensor's minimum and maximum value as well as the number of ranges.

Request. The request table realizes the request log by tracking requests. A request is the unique combination of a requested fragment and the requesting data user. The request log tracks the watermark number, which was embedded as a watermark, and logs the timestamps of a request in an array which may be helpful if, in addition, the result of watermark detection is manually investigated.

5.3. Services

The services are realized by Java classes and can access the database to store and retrieve data. The simplified class diagram is shown in Figure 17. In order to improve readability, only the public methods without their whole method signatures are shown.

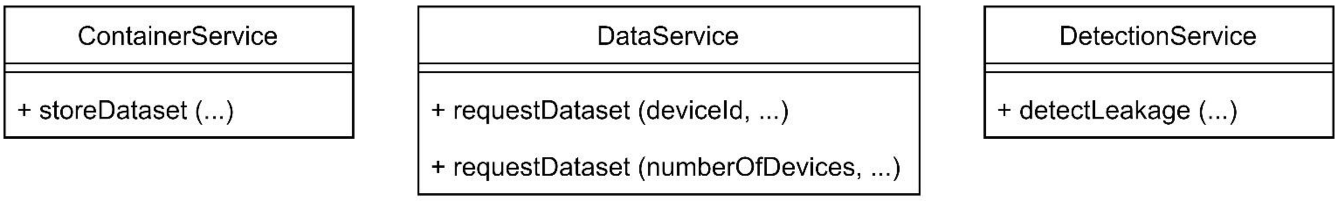


Figure 17: Simplified Class Diagram of the Services

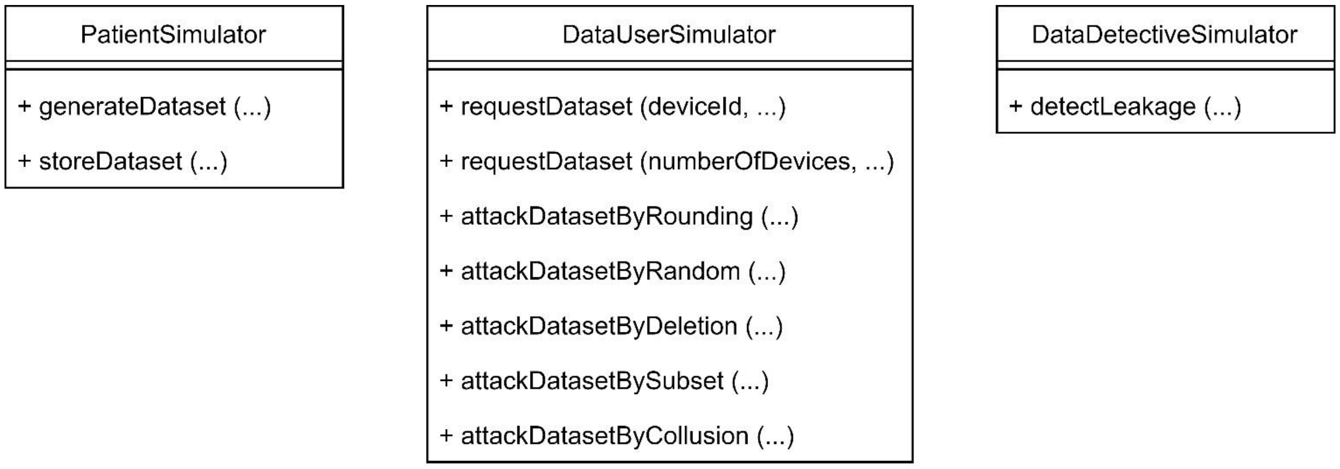
Container Service. The container service provides a method to store a dataset. The dataset is read from the file system and fragmented based on the fragmentation criteria. In this thesis, it is assumed that the fragmentation criteria are always sensor and day. After the fragmentation, each fragment is stored together with a random secret key in the database.

Data Service. The data service provides two methods to request a dataset of a certain time period. The first method is used to request data of a single device; the second method is used to request data of multiple devices. The requested fragments are retrieved from the database and watermarked. The watermarked fragments are combined to a watermarked dataset which is provided in the file system.

Detection Service. The detection service provides a method to detect data leakage of a suspicious dataset. The suspicious dataset is read from the file system and fragmented based on the fragmentation criteria sensor and day. After that, watermark detection is performed and, finally, a leakage report is provided in the file system.

5.4. Simulators

The simulators are realized by Java classes and represent roles using the services. The simplified class diagram is shown in Figure 18. In order to improve readability, only the public methods without their whole method signatures are shown.


PatientSimulator
+ generateDataset (...)
+ storeDataset (...)
DataUserSimulator
+ requestDataset (deviceId, ...)
+ requestDataset (numberOfDevices, ...)
+ attackDatasetByRounding (...)
+ attackDatasetByRandom (...)
+ attackDatasetByDeletion (...)
+ attackDatasetBySubset (...)
+ attackDatasetByCollusion (...)
DataDetectiveSimulator
+ detectLeakage (...)
Figure 18: Simplified Class Diagram of the Simulators

Patient Simulator. The patient simulator provides a method to generate a dataset and a method to store a dataset. The generation method produces a dataset which consists of pseudo random measurements of continuous blood glucose in mmol/L of a given period of time. It should be noted that timestamps of generated fragments from the same day but from different devices are always equal. The generated dataset is provided in the file system. The store method executes the container service to store a given dataset.

Data User Simulator. The data user simulator provides two methods to request a dataset and five methods to attack a dataset. The first request method is used to request data of a single device; the second request method is used to request data of multiple devices. The request methods execute the corresponding methods from the data service. An attack method reads a dataset from the file system, attacks it based on the selected attack method and its configuration, and provides the attacked dataset in the file system.

Data Detective Simulator. The data detective simulator provides a method to detect leakage of a given suspicious dataset by executing the corresponding method from the detection service.

5.5. Interfaces

The user interfaces are realized by Java classes and configure the simulators. The program user interface is used when executing the Java program in a development environment. The console user interface is used when executing the jar file enabling the use of batch files to run many operations with different configurations. The console user interface is shown in Figure 19 and is explained in more detail below.

The *reset* section provides commands to remove all records from the selected database table such that testing with different database content is possible. Furthermore, this section provides commands to reset the file system before running new tests by removing produced files and the log.
The *set* section inserts a usability constraint for continuous blood glucose in mmol/L with a sensor's minimum value of 0 mmol/L and a sensor's maximum value 55 mmol/L as defined by the data model of Tidepool[4]. The configuration of a usability constraint can be changed using different maximum error and number of ranges.

[4] Tidepool Data Model: http://developer.tidepool.org/data-model/

```
Commands
-reset -table -fragment
-reset -table -request
-reset -table -usability_constraint
-reset -files
-reset -log

-set -usability_constraint [maximumError] [numberOfRanges]

-generate [deviceId] [from] [to] [seed]

-store -random [datasetName]
-store -one [datasetName]

-request -patient [dataUserId] [deviceId] [from] [to]
-request -patients [dataUserId] [numberOfDevices] [from] [to]
-attack -deletion [datasetName] [frequency]
-attack -random [datasetName] [maximumError] [seed]
-attack -rounding [datasetName] [decimalDigit]
-attack -subset [datasetName] [startIndex] [endIndex]
-attack -collusion [datasetName1] ... [datasetNameN]

-detect [datasetName] [fragmentSimilarityThreshold]
            [watermarkSimilarityThreshold] [numberOfColluders]

Configurable Parameters
[dataUserId]:                  data user identifier as integer
[datasetName]:                 dataset name as string
[decimalDigit]:                decimal digit as integer
[deviceId]:                    device identifier as string
[endIndex]:                    end index as integer
[fragmentSimilarityThreshold]: fragment similarity threshold as double
[frequency]:                   frequency as integer
[from]:                        start date as string with the format [yyyy-MM-dd]
[maximumError]:                maximum error as double
[numberOfColluders]:           number of colluders as integer
[numberOfDevices]:             number of devices as integer
[numberOfRanges]:              number of ranges as integer
[seed]:                        seed as integer
[startIndex]:                  start index as integer
[to]:                          end date as string with the format [yyyy-MM-dd]
[watermarkSimilarityThreshold]: watermark similarity threshold as double
```

Figure 19: Console User Interface

The *generate* section executes the patient simulator to generate a dataset of a device over a certain period of time with the measurement values being pseudo randomly selected based on a seed.
The *store* section executes the patient simulator to store a dataset in the database. The secret key assigned to the fragments can be generated randomly or can simply be 1 such that experiments can be reproduced easily.
The *request* section executes the data user simulator to request a dataset of a single device or of multiple devices over a certain period of time.
The *attack* section executes the data user simulator to attack a dataset by the selected method and its configuration. The deletion attack removes measurements based on a frequency; the random attack pseudo randomly changes values based on a maximum error and seed; the rounding attack rounds

values based on a decimal digit; the subset attack selects a subset of measurements from fragments based on a start and an end index; the collusion attack creates a new dataset based on multiple watermarked datasets.
The *detect* section executes the data detective simulator to detect leaking data users of a given dataset based on similarity thresholds and the number of colluders. In the end, the data types of the configurable parameters are provided.

In this chapter, the implementation of the proof-of-concept prototype was presented by introducing its platform and the components of the main levels. In the next chapter, the implemented prototype is evaluated regarding its requirements and design goals.

6. Evaluation

This chapter evaluates the implemented proof-of-concept prototype which is based on the introduced watermarking approach regarding the requirements and design goals. In the end, the result of the evaluation is summarized.

6.1. Evaluation of Requirements based on the Problem Statement

This section provides the evaluation of the prototype regarding requirements based on the problem statement.

PS-1: The prototype shall be based on watermarking. The prototype is based on watermarking if the prototype performs watermarking according to the introduced definition of watermarking by Cox et al. [3]. Cox et al. [3] define watermarking as "… *the practice of imperceptibly altering a work to embed a message about that work*". Furthermore, Cox et al. [3] name three important attributes that distinguish watermarking from other techniques, namely, imperceptibility, inseparability and the ability of a watermark to undergo the same transformations as the watermarked data.

Imperceptibility. The imperceptibility of a watermark depends on the usability constraint which limits the maximum error and the error distribution of watermark generation. We assume that a watermark is imperceptible if the watermarked fragment is subjectively considered as visually similar to the original one. Figure 20 compares an excerpt of a fragment watermarked using a loose usability constraint with an excerpt of the same fragment watermarked using a tight usability constraint. The use of a loose usability constraint results in clear visible differences between original and watermarked values, but the watermark may be perceived as imperceptible without comparing it to the original fragment. The use of a tight usability constraint results in no clearly visible differences between original and watermarked values anymore.
Since the imperceptibility of a watermark depends on the usability constraint, an appropriate usability constraint is required. The selection of appropriate usability constraints may be one of the major challenges for data providers, but also improves the adaptability of the prototype to different application areas.

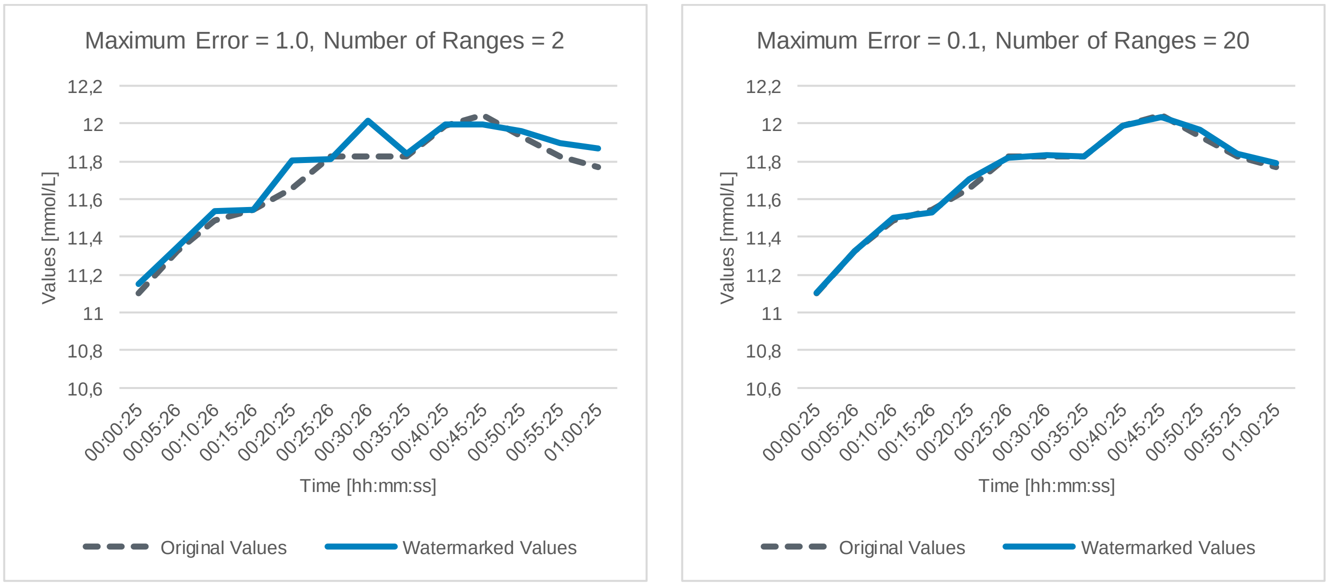


Figure 20: Visual Watermark Imperceptibility using a loose and tight Usability Constraint

Inseparability. The watermark is inseparable from the data due to being embedded in the measurement values of a fragment. The only way to separate the watermark from the data is by removing the measurement values which, in the case of medical sensor data, results in useless data.

Same Transformations. The watermark undergoes the same transformations as the data due to being embedded in the measurement values of a fragment. We assume that transformations typically target the values of measurements, which include the watermark.

In summary, the watermark can be imperceptible depending on the usability constraint, it is inseparable and it, typically, undergoes the same transformations. Furthermore, according to the above mentioned definition of watermarking by Cox et al. [3], we can state that the prototype imperceptibly alters a fragment to embed a message about the receiver of that fragment. This requirement is fulfilled because the prototype performs watermarking analogously to the definition of watermarking by Cox et al. [3].

PS-2: The prototype shall be applicable to medical sensor data. The prototype is applicable to medical sensor data if it can watermark the example medical sensor data described in Section 1.2.
The watermarking approach, which is the basis for the prototype, is designed to be applied to medical sensor data due to watermarking of measurement values while considering preserving diagnostic semantics. We used the prototype to request all fragments from the example medical sensor data and received a dataset with all the fragments being watermarked, proofing that the prototype is able to watermark medical sensor data.
This requirement is fulfilled because the prototype can watermark the example medical sensor data described in Section 1.2.

PS-3: The prototype shall detect data leakage. The prototype detects data leakage if, given a suspicious dataset, the prototype can identify whether the dataset originates from the database and, if so, who leaked the data.
These questions are answered per fragment and then combined to an overall answer for the dataset. Whether a suspicious fragment originates from the database is determined by the fact that its fragment similarity with an original fragment from the database exceeds a user-defined threshold. If this is the case, data users are suspects of leaking that fragment if their associated watermarks are considered as matching watermarks. Matching watermarks have a watermark similarity with the extracted watermark exceeding a user-defined threshold. The combination of all matching watermarks from the suspicious fragments reliably identifies the leaking data user of the suspicious dataset. Even in case of malicious attacks the leaking data users may be identified by watermark combinations.

Figure 21 provides an example watermark detection report which was produced by the prototype. In this example, the suspicious dataset consists of two fragments. The first fragment was requested by data user 1 and 2, and the second fragment was requested by data user 1 only. We assume that data user 1 leaked the requested fragments without attacking the data maliciously.
The first section of the report indicates the parameters used for watermark detection. In this example, the user-defined thresholds for fragment and watermark similarity are only 1 %. The number of colluders indicates the maximum number of watermarks to include in watermark combinations.
The second section shows the result of watermark detection for each fragment of the suspicious dataset. It provides the meta data of the suspicious fragment and its matching fragment as well as their fragment similarity. Furthermore, the identified matching watermarks are provided as watermark similarities together with their associated data users. Due to the number of colluders being configured as 2, watermark combinations of maximum two watermarks are also matched.

In the last section, the results of watermark detection from each suspicious fragment are combined to compute the probability of dataset leakage for each data user with at least one matching watermark.

This requirement is fulfilled because the prototype can identify whether a dataset originates from the database and who leaked the data, as shown in the example watermark detection report of Figure 21.

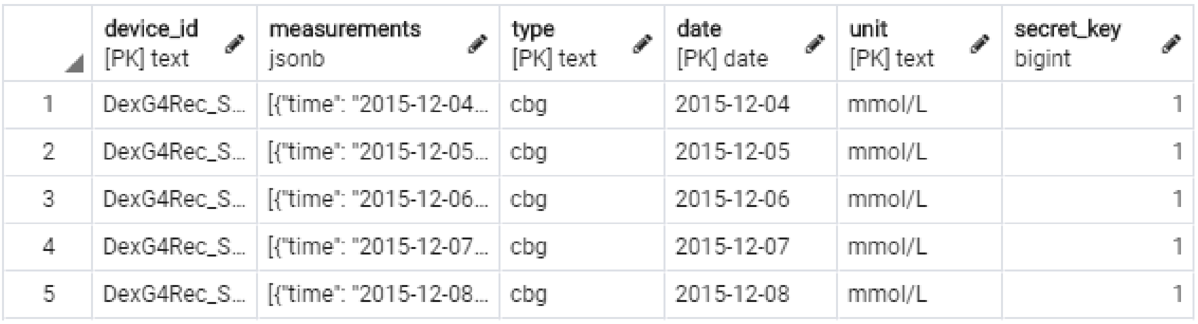
```
watermark detection report
fragment similarity threshold:  0.01
watermark similarity threshold: 0.01
number of colluders:            2

[1] fragment leakage
suspicious fragment:            DexG5MobRec_SM64305440    cbg    mmol/L 2017-02-04
matching original fragment:     DexG5MobRec_SM64305440    cbg    mmol/L 2017-02-04
matching fragment similarity:   0.9987
matching watermarks:
             probability of 1.0000     by data user [1]
             probability of 0.9432     by data user [1, 2]
             probability of 0.8865     by data user [2]
[2] fragment leakage
suspicious fragment:            DexG5MobRec_SM64305440    cbg    mmol/L 2017-02-05
matching original fragment:     DexG5MobRec_SM64305440    cbg    mmol/L 2017-02-05
matching fragment similarity:   0.9988
matching watermarks:
             probability of 1.0000     by data user [1]

dataset leakage
             probability of 1.0000     by data user [1]
             probability of 0.4716     by data user [1, 2]
             probability of 0.4433     by data user [2]
```

Figure 21: Example Watermark Detection Report

6.2. Evaluation of Requirements based on the Solution Idea

This section provides the evaluation of the prototype regarding requirements based on the solution idea. It should be noted that the content of tables providing information about watermarks is based on output from the prototype. In addition, the parameters which were used to configure the prototype are provided in the first line.

SI-1: The prototype shall implement the use of dataset fragments. The prototype uses dataset fragments if measurements from the same sensor and from the same day are stored together in the database.
The container service fragments measurements by collecting measurements of the same device, type, unit and day. The resulting fragments of a dataset are stored in the fragment table of the database. Figure 22 shows an excerpt from the fragment table after storing the example medical sensor data, indicating that the data is fragmented and stored based on sensor and day.
This requirement is fulfilled because the prototype stores measurements from the same sensor and from the same day together in the database.



	device_id [PK] text	measurements jsonb	type [PK] text	date [PK] date	unit [PK] text	secret_key bigint
1	DexG4Rec_S...	[{"time": "2015-12-04...	cbg	2015-12-04	mmol/L	1
2	DexG4Rec_S...	[{"time": "2015-12-05...	cbg	2015-12-05	mmol/L	1
3	DexG4Rec_S...	[{"time": "2015-12-06...	cbg	2015-12-06	mmol/L	1
4	DexG4Rec_S...	[{"time": "2015-12-07...	cbg	2015-12-07	mmol/L	1
5	DexG4Rec_S...	[{"time": "2015-12-08...	cbg	2015-12-08	mmol/L	1

Figure 22: Example Excerpt from the Fragment Table

SI-2: The prototype shall implement the proposed watermark embedding. The prototype implements the proposed watermark embedding the sub-requirements are fulfilled.

SI-2a: The watermark generation shall be configurable by a requested fragment. The watermark generation is configurable by a requested fragment if watermark generation produces different watermarks for different fragments.
The watermark generation is configurable by measurement values of a requested fragment because watermark generation produces a watermark which preserves the value structure of measurement values. Table 6 lists the first five errors of two different watermarks which have the exact same parameters for watermark generation except their measurement values.
The watermark generation is also configurable by the secret key of a requested fragment because the PRNG for error selection is seeded, among others, with the secret key. Table 7 lists the first five errors of two different watermarks which have the exact same parameters for watermark generation except their secret key.
This sub-requirement is fulfilled because watermark generation produces different watermarks for different fragments.

Secret Key = 1, Watermark Number = 1, Maximum Error = 0.5, Number of Ranges = 10		
Error	Fragment 2017-02-04	Fragment 2017-02-05
1	0.009814277224974	0.022101249328929
2	0.005019538581845	0.003346359054561
3	0.021052543106296	0.189636992530182
4	-0.023823737696820	-0.345723137329108
5	0.162725738541118	0.488600031333992

Table 6: Example Errors of Watermarks using different Measurements

Fragment 2017-02-04, Watermark Number = 1, Maximum Error = 0.5, Number of Ranges = 10		
Error	Secret Key = 1	Secret Key = 2
1	0.009814277224974	0.146309434848484
2	0.005019538581845	0.121608522192820
3	0.021052543106296	0.015272552704122
4	-0.023823737696820	-0.010195062634422
5	0.162725738541118	-0.042314868612707

Table 7: Example Errors of Watermarks using different Secret Keys

SI-2b: The watermark generation shall be configurable by a usability constraint. The watermark generation is configurable by a usability constraint if watermark generation produces different watermarks using different usability constraints and if watermarks are able to preserve the metrics by Battelino et al. [6].

The watermark generation is configurable by the usability constraint because the size and distribution of errors is limited by the maximum error, the sensor's minimum and maximum values as well as the number of ranges. Table 8 lists the first five errors of two different watermarks which have the exact same parameters for watermark generation except their usability constraint.
Table 9 provides example deviations of watermarked fragments from the metrics by Battelino et al. [6] using a loose and a tight usability constraint. Using the loose usability constraint results in little deviations in the mean, glucose management indicator and glycemic variability, while using the tight usability constraint results in literally no deviations in these metrics.
The metrics of Table 9 indicating the time within a range are computed by counting the number of measurements within a range because measurements are typically performed every 5 minutes. In both cases, using the loose and tight usability constraint result in high relative but only little absolute deviations considering that there are 286 measurements in this fragment. These deviations caused by watermarking may be considered negligible assuming that they only slightly affect the diagnosis of a patient if the patient remained approximately for 5 minutes more in a certain range. In addition, further tightening the usability constraint may also reduce these deviations.
This sub-requirement is fulfilled because watermark generation produces different watermarks using different usability constraints and watermarks are able to mostly preserve the metrics by Battelino et al. [6].

Fragment 2017-02-04, Watermark Number = 1, Secret Key = 1		
Error	Maximum Error = 1.0, Number of Ranges = 2	Maximum Error = 0.1, Number of Ranges = 20
1	0,049071386	0,002210125
2	0,025097693	0,001507166
3	0,049755236	0,010526272
4	-0,002630107	-0,015551044
5	0,147538934	0,048860003

Table 8: Example Errors of Watermarks using different Usability Constraints

Metrics		Original	Maximum Error = 1.0, Number of Ranges = 2		Maximum Error = 0.1, Number of Ranges = 20	
			Watermarked	Δ	Watermarked	Δ
Mean Glucose		10.2609	10.2762	0.15 %	10.2607	0.00 %
Glucose Management Indicator		60.9965	61.0685	0.12 %	60.9954	0.00 %
Glycemic Variability 1		5.4146	5.4109	-0.07 %	5.4146	0.00 %
Glycemic Variability 2		15.1072	15.1415	0.23 %	15.1067	0.00 %
Time above Range:	> 13.9	54	57	5.56 %	54	0.00 %
Time above Range:	10.1 – 13.9	78	74	-5.13 %	77	-1.28 %
Time in Range:	3.9 – 10.0	141	142	0.71 %	140	-0.71 %
Time below Range:	3.0 – 3.8	6	8	33.33 %	7	16.67 %
Time below Range:	< 3.0	3	3	0.00 %	2	-33.33 %

Table 9: Example Deviations from Metrics using different Usability Constraints

SI-2c: The watermark generation shall be configurable by a watermark number. The watermark generation is configurable by a watermark number if watermark generation produces different watermarks using different watermark numbers.

The watermark generation is configurable by a watermark number because the PRNG for error selection is seeded, among others, with the watermark number. Table 10 lists the first five errors of two different watermarks which have the exact same parameters for watermark generation except their watermark number.
This sub-requirement is fulfilled because watermark generation produces different watermarks using different watermark numbers.

Fragment 2017-02-04, Secret Key = 1, Maximum Error = 0.5, Number of Ranges = 10		
Error	Watermark Number = 1	Watermark Number = 2
1	0.009814277224974	0.030333447467151
2	0.005019538581845	-0.122482698455908
3	0.021052543106296	0.040056718675180
4	-0.023823737696820	0.033182217222552
5	0.162725738541118	0.022100688158746

Table 10: Example Errors of Watermarks using different Watermark Numbers

SI-2d: The watermark generation shall generate a matching watermark. The watermark generation generates a matching watermark if the result of watermark generation is a matching watermark.
The watermark generation produces a sequence of matching errors with the same length as the requested fragment, i.e. a matching watermark, based on the fragment, its usability constraint and the watermark number. Table 11 again shows the example watermark from Figure 10 which is a matching watermark of the example fragment from Figure 3 is and which is generated by the prototype.
This sub-requirement is fulfilled because the result of watermark generation is a matching watermark.

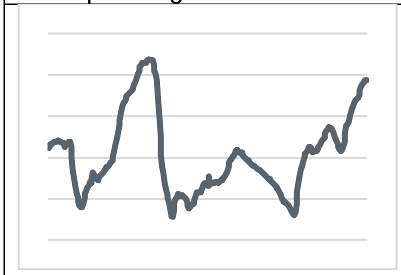
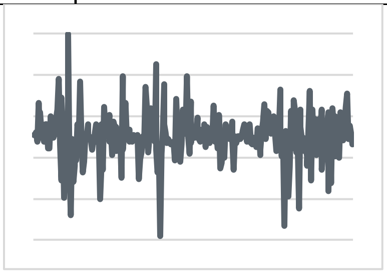
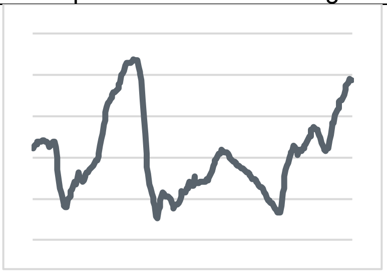
Figure 3 Example Fragment	Figure 10 Example Watermark	Figure 11 Example Watermarked Fragment
		

Table 11: Example Watermark Embedding

Fragment 2017-02-04, Secret Key = 1, Watermark Number = 1, Maximum Error = 0.5, Number of Ranges = 10			
Measurement	Requested Fragment	Watermark	Watermarked Fragment
1	11.101495982091066	0.009814277224974	11.111310259316040
2	11.323525901732888	0.005019538581845	11.328545440314733
3	11.490048341464254	0.021052543106296	11.511100884570550
4	11.545555821374709	-0.023823737696820	11.521732083677889
5	11.656570781195620	0.162725738541118	11.819296519736738

Table 12: Example Embedding of a Watermark

SI-2e: The generated watermark shall be embedded. The generated watermark is embedded if the result of embedding is a watermarked fragment.
The generated watermark is embedded into a requested fragment by adding the errors of the watermark to the corresponding measurement values which results in a watermarked fragment. Table 12 provides an example for embedding by listing five measurement values of the requested fragment from Figure 3,

their corresponding error from the matching watermark from Figure 10 and their sum as new measurement values of the watermarked fragment from Figure 11.
This sub-requirement is fulfilled because the result of embedding is a watermarked fragment.

Summarizing SI-2, watermark generation is configurable by a requested fragment, a usability constraint and a watermark number. Furthermore, the watermark generation generates a matching watermark which is embedded in the requested fragment. This requirement is fulfilled because all its sub-requirements are fulfilled.

SI-3: The prototype shall implement the proposed watermark detection. The prototype implements the proposed watermark detection if the sub-requirements are fulfilled.

SI-3a: The fragment similarity search shall identify a matching fragment. The fragment similarity search identifies a matching fragment if the result of fragment similarity search is a matching fragment.
The fragment similarity search computes the fragment similarities between a suspicious fragment and each relevant original fragment based on a matching measurements analysis. If the fragment with the highest similarity exceeds a user-defined threshold, the matching fragment is found. Figure 23 provides a snippet from the example watermark detection report from Figure 21 which shows metadata of a suspicious fragment and its identified matching fragment together with their fragment similarity.
This sub-requirement is fulfilled because the result of fragment similarity search is a matching fragment.

```
[1] fragment leakage
suspicious fragment:           DexG5MobRec_SM64305440    cbg    mmol/L 2017-02-04
matching original fragment:    DexG5MobRec_SM64305440    cbg    mmol/L 2017-02-04
matching fragment similarity:  0.9987
```

Figure 23: Example Identification of a Matching Fragment

SI-3b: The embedded watermark shall be extracted. The embedded fragment is extracted if the result of extraction is a watermark.
The embedded watermark is extracted by subtracting each measurement value of the matching fragment from the corresponding measurement value of the suspicious fragment. The resulting sequence of differences (or errors) is the extracted watermark. Table 13 provides an extraction example by listing five measurement values of the watermarked fragment from Figure 11, their corresponding measurement value of the matching fragment from Figure 3 and their differences as the extracted watermark from Figure 10.
This sub-requirement is fulfilled because the result of extraction is a watermark.

Fragment 2017-02-04, Secret Key = 1, Watermark Number = 1, Maximum Error = 0.5, Number of Ranges = 10			
Measurement	Watermarked Fragment	Matching Fragment	Extracted Watermark
1	11.111310259316040	11.101495982091066	0.009814277224974
2	11.328545440314733	11.323525901732888	0.005019538581845
3	11.511100884570550	11.490048341464254	0.021052543106296
4	11.521732083677889	11.545555821374709	-0.023823737696820
5	11.819296519736738	11.656570781195620	0.162725738541118

Table 13: Example Extraction of an Embedded Watermark

RSI-3c: The already embedded watermarks shall be re-generated. The already embedded watermarks are re-generated if the result of watermark re-generation is a set of already embedded watermarks.

The already embedded watermarks are re-generated based on the matching fragment, its usability constraint and the watermark numbers which were already used for the matching fragment. For each watermark number, watermark generation from watermark embedding is performed. The set of re-generated watermarks can optionally be enriched by watermark combinations.
Figure 24 provides a snippet from the example watermark detection report of Figure 21 which shows the following two things. First, two watermarks and one combination of two watermarks have been identified as matching watermarks proofing that at least two watermarks were re-generated. Second, a matching watermark similarity of 100 % between the re-generated watermark of data user 1 and the extracted watermark (associated to data user 1) proves that watermark re-generation was performed correctly.
This sub-requirement is fulfilled because the result of watermark re-generation is a set of already embedded watermarks.

```
matching watermarks:
             probability of 1.0000     by data user [1]
             probability of 0.9432     by data user [1, 2]
             probability of 0.8865     by data user [2]
```

Figure 24: Example Watermark Re-Generation and Example Matching Watermarks

SI-3d: The watermark similarity search shall identify matching watermarks. The watermark similarity search identifies matching watermarks if the result of watermark similarity search are matching watermarks.
The watermark similarity search computes the watermark similarities between an extracted watermark and each watermark from the set of re-generated watermarks. If a watermark similarity exceeds a user-defined threshold, a matching watermark is found. Figure 24 provides a snippet from the example watermark detection report of Figure 21 which shows that three matching watermarks were identified by indicating their watermark similarity together with their associated data user.
This sub-requirement is fulfilled because the result of watermark similarity search are matching watermarks.

SI-3e: The leaker identification shall identify potential data leakers. The leaker identification identifies potential data leakers if the result of leaker identification is a list of potential data leakers together with their leakage probabilities.
The leaker identification computes the average watermark similarities of each data user who is associated with at least one identified matching watermark in the dataset. The average watermark similarity indicates the probability of a data user to be a data leaker. Figure 25 provides a snippet from the example watermark detection report of Figure 21 which shows three potential data leakers of a dataset together with their leakage probabilities.
This sub-requirement is fulfilled because the result of leaker identification is a list of potential data leakers together with their leakage probabilities.

```
dataset leakage
             probability of 1.0000     by data user [1]
             probability of 0.4716     by data user [1, 2]
             probability of 0.4433     by data user [2]
```

Figure 25: Example Identification of Potential Data Leakers

Summarizing SI-3, fragment similarity search identifies a matching fragment, the embedded watermark is extracted, already embedded watermarks are re-generated, watermark similarity search identifies matching watermarks and leaker identification identifies potential data leakers. This requirement is fulfilled because all its sub-requirements are fulfilled.

6.3. Evaluation of Requirements and Design Goals based on Watermarking Properties

This section provides the evaluation of the prototype regarding requirements and design goals based on watermarking properties named by Cox et al. [3].

WP-1: The embedding effectiveness shall be 100%. The embedding effectiveness is 100 % if the probability of successfully embedding a watermark into a randomly selected fragment is 100 %.
The prototype embeds a watermark into every fragment of a requested dataset. We used the prototype to request all fragments from the example medical sensor data and received a dataset with all the fragments being watermarked.
This requirement is fulfilled because the probability of successfully embedding a watermark into a randomly selected fragment is 100 %.

WP-2: The fidelity should be preferably high. The fidelity is measured by the similarities between an original fragment and its watermarked fragments with the similarities being computed by fragment similarity search of watermark detection.
The fidelity depends on the usability constraint which limits the maximum error and the error distribution of watermark generation. Table 14 lists the fragment similarities of a fragment that has been watermarked, using three different usability constraints and five different watermark numbers each. The tighter the usability constraint, the more similar are watermarked fragments to the original fragment.
The fidelity can be considered high because even using a loose usability constraint the fragment similarities are > 99.50 %.

Fragment 2017-02-04, Secret Key = 1			
Watermark Number	Maximum Error = 1.0, Number of Ranges = 2	Maximum Error = 0.5, Number of Ranges = 10	Maximum Error = 0.1, Number of Ranges = 20
1	99.67 %	99.87 %	99.98 %
2	99.70 %	99.87 %	99.98 %
3	99.67 %	99.85 %	99.97 %
4	99.65 %	99.86 %	99.98 %
5	99.69 %	99.86 %	99.98 %

Table 14: Example Fragment Similarities using different Usability Constraints

WP-3: The data payload should be as high as necessary. The data payload is measured by the number of bits encoded in a watermark.
The watermark as a sequence of errors with each error having 15 decimal digits, and with the same length as the fragment has a considerably high size. Nevertheless, the information which is stored in the database and which is encoded as watermark is the watermark number. Table 15 lists different numbers of watermarks and their data payload as number of bits. With an increasing number of requests on a fragment, the watermark number and therefore its data payload increases. Even using a small integer of 2 bytes for the watermark number, 65 536 different watermarks can be encoded.
It should be noted that we assume that with an increasing number of watermarks, the distinctiveness between watermarks decreases. The investigation of this assumption, however, is out-of-scope of this thesis.

The data payload can be considered as high as necessary because the data payload is only 2 bytes for 65 536 different watermarks.

Number of Watermarks	Number of Bits
2	1
256	8
65 536	16

Table 15: Example Data Payload

WP-4: The detection shall be informed. The detection is informed if original fragments are utilized during watermark detection.
The watermark detection uses original fragments for fragment similarity search to identify matching fragments and uses these original matching fragments to extract an embedded watermark. Informed detection is already shown by the fact that requirement SI-3 is fulfilled, in particular the sub-requirements SI-3a and SI-3b.
We also prove the contrary that without original fragments, watermark detection already fails during fragment similarity search. Figure 26 shows an excerpt of a watermark detection report in which the prototype could not identify a matching fragment because the database was empty i.e. no original fragments were available.
This requirement is fulfilled because original fragments are utilized during watermark detection.

```
[1] fragment leakage
suspicious fragment:            DexG5MobRec_SM64305440    cbg    mmol/L 2017-02-04
no matching fragment detected
```

Figure 26: Example Detection Report using Blind Detection

WP-5: The false positive rate should be preferably low. The false positive is measured by the frequency of detecting a matching watermark in unwatermarked fragments.
The frequency of false positive detection depends on the number of relevant fragments in the database and the number of watermarks for each relevant fragment. We assume that a false positive detection means very high fragment similarity and 100% watermark similarity. This design goal is not evaluated using the prototype because the result might not be representative and only true for this specific setting. Instead, this design goal is evaluated based on the following theoretical considerations.

The frequency of an unwatermarked fragment consisting of exactly the same measurement values as a watermarked fragment can be computed using different settings regarding the content of the database. In order to measure the false positive rate theoretically, the following assumptions are made regarding measurements of continuous blood glucose. The measurements are performed every 5 minutes; therefore, a fragment must consist of 288 measurements. The value of a measurement is between 0 and 55 mmol/L with only 4 decimal digits being relevant for watermark similarity. Furthermore, the value change between two measurements is between 0 and 1 mmol/L. Based on these assumptions, there are $55*10^4$ possible values for the first measurement and $287*20^4$ possible values for the following measurements. This results in $2.5256*10^{13}$ possible value combinations for a fragment. For the relevant fragments in the database, we assume that they (and their watermarked versions) do not overlap.
Table 16 provides the frequency of false positives in different settings based on the Laplace Probability and $2.5256*10^{13}$ possible value combinations. The settings consist of the number of relevant fragments in the database and the number of watermarks used for each relevant fragment. In a typical database setting, a false positive is very unlikely especially if the number of relevant fragments is limited by any criteria. Furthermore, limiting the number of different watermarks (e.g. to ensure maximally different watermarks) may also improve the false positive rate.

These considerations are, however, limited to the following facts. First and foremost, these considerations are only true for the assumptions regarding measurements of continuous blood glucose. In addition, we only consider false positives with watermark similarity of 100 % and that every relevant fragment has the same timestamps as the unwatermarked fragment. Furthermore, only fragments with exactly 288 measurements are considered, while a fragment can also consist of fewer. Finally, it should be noted that we theoretically specified false positive detection of a single fragment which markedly differs from the false positive detection of a whole dataset.

In summary, the false positive rate can be considered very low because at least a billion different fragments with 256 different watermarks each are necessary for a false positive rate > 1 %.

Number of Relevant Fragments	Number of Watermarks	Frequency
10^6	1	0.000004 %
10^6	256	0.001014 %
10^9	1	0.003959 %
10^9	256	1.013621 %

Table 16: Theoretical False Positive Rate

WP-6: The robustness should be preferably high. The robustness is measured together with the closely related WP-7.

WP-7: The security should be preferably high. The security is measured by the sub-design goals. The sub-design goals measure the security against different attack methods. For this purpose, we watermarked a single fragment, then attacked it using different attacks with different configurations. The watermark detection determines the fragment similarity and watermark similarities of five different watermarks. The results of watermark detection for each attack are shown in a table with each line per configuration. The first column indicates the configuration of the attack method, the second column shows the matching fragment similarity and the remaining columns list the watermark similarities. It should be noted that the watermark similarities of the embedded watermark are shown in bold.

Fragment 2017-02-04, Secret Key = 1, Watermark Number = 1 … 5, Maximum Error = 0.5, Number of Ranges = 10						
Decimal Digit	Matching Fragment Similarity	Matching Watermark Similarity				
		1	2	3	4	5
4	99.87 %	**100.00 %**	88.65 %	88.10 %	89.06 %	88.86 %
3	99.87 %	**99.98 %**	88.65 %	88.09 %	89.06 %	88.86 %
2	99.87 %	**99.77 %**	88.65 %	88.08 %	89.04 %	88.88 %
1	99.86 %	**97.53 %**	88.22 %	87.62 %	88.80 %	88.71 %
0	99.53 %	**76.06 %**	72.61 %	70.94 %	72.35 %	72.02 %

Table 17: Example Security against Rounding Value Attacks

WP-7a: The prototype should be secure against rounding value attacks. The security against rounding value attacks is measured by watermark similarities of a watermarked fragment which was target of rounding value attacks using different configurations.
The rounding value attack rounds values of measurements based on a decimal digit, eventually distorting the watermark. Table 17 provides an example security report against rounding value attacks using different configurations. The reliability of the watermark decreases notably when rounding the measurements to whole numbers, but this may also exceed the usability constraint. Nevertheless, even

if rounding the measurements to whole numbers, the watermark similarity of the embedded watermark is markedly higher than the watermark similarities of other watermarks.

WP-7b: The prototype should be secure against random value attacks. The security against random value attacks is measured by watermark similarities of a watermarked fragment which was target of random value attacks using different configurations.
The random value attack changes the values of measurements randomly based on a maximum error, eventually distorting the watermark. Table 18 provides an example security report against random value attacks using different configurations. The reliability of the watermark decreases if the maximum error increases. Nevertheless, even changing the values by a maximum of 0.5, the watermark similarity of the embedded watermark is markedly higher than the watermark similarities of other watermarks.

Fragment 2017-02-04, Secret Key = 1, Watermark Number = 1 … 5, Maximum Error = 0.5, Number of Ranges = 10						
Maximum Error	Matching Fragment Similarity	Matching Watermark Similarity				
		1	2	3	4	5
0.1	99.83 %	**94.92 %**	87.34 %	86.85 %	87.36 %	87.51 %
0.2	99.76 %	**89.83 %**	84.51 %	83.78 %	84.24 %	84.47 %
0.3	99.68 %	**84.75 %**	80.87 %	80.00 %	80.33 %	80.51 %
0.4	99.59 %	**79.66 %**	76.67 %	75.88 %	75.98 %	76.23 %
0.5	99.50 %	**74.58 %**	72.16 %	71.40 %	71.29 %	71.60 %

Table 18: Example Security against Random Value Attacks

WP-7c: The prototype should be secure against deletion attacks. The security against deletion attacks is measured by watermark similarities of a watermarked fragment which was target of deletion attacks using different configurations.
The deletion attack removes measurements based on a frequency. Table 19 provides an example security report against deletion attacks using different configurations. The watermark similarity of the embedded watermark is not affected (100 %) because the matching measurements analysis can identify all matching measurements due to its time dependence.

Fragment 2017-02-04, Secret Key = 1, Watermark Number = 1 … 5, Maximum Error = 0.5, Number of Ranges = 10						
Frequency	Matching Fragment Similarity	Matching Watermark Similarity				
		1	2	3	4	5
5	99.86 %	**100.00 %**	88.39 %	87.64 %	88.47 %	89.11 %
4	99.87 %	**100.00 %**	88.86 %	87.74 %	89.22 %	88.65 %
3	99.86 %	**100.00 %**	88.73 %	88.17 %	88.96 %	88.85 %
2	99.86 %	**100.00 %**	88.01 %	88.22 %	89.20 %	88.79 %

Table 19: Example Security against Deletion Attacks

WP-7d: The prototype should be secure against subset selection attacks. The security against subset selection attacks is measured by watermark similarities of a watermarked fragment which was target of subset selection attacks using different configurations.
The subset selection attack selects a subset of measurements from each fragment based on a start and an end index. Table 20 provides an example security report against subset selection attacks using different configurations. The result is basically the same as in WP-7c, due to the time dependent matching measurements analysis, the watermark similarity of the embedded watermark is not affected (100 %).

Fragment 2017-02-04, Secret Key = 1, Watermark Number = 1 … 5, Maximum Error = 0.5, Number of Ranges = 10						
Index	Matching Fragment Similarity	Matching Watermark Similarity				
		1	2	3	4	5
100 – 159	99.87 %	**100.00 %**	89.97 %	89.26 %	89.35 %	88.00 %
100 – 147	99.85 %	**100.00 %**	89.43 %	87.84 %	88.39 %	87.07 %
100 – 135	99.84 %	**100.00 %**	88.29 %	85.68 %	87.62 %	86.51 %
100 – 123	99.81 %	**100.00 %**	85.88 %	84.41 %	89.09 %	85.35 %
100 – 111	99.79 %	**100.00 %**	82.87 %	88.09 %	86.96 %	85.86 %

Table 20: Example Security against Subset Selection Attacks

WP-7e: The prototype should be secure against mean collusion attacks. The security against mean collusion attacks is measured by watermark similarities of a colluded fragment which was target of mean collusion attacks using different configurations.
The mean collusion attack creates a new dataset based on mean of values of differently watermarked measurements. The watermark detection of a mean colluded dataset is always 100 % successful if the correct number of colluders is given to the algorithm due to being able to arbitrary combine (mean) watermarks.
We are, however, more interested in the watermark similarities of colluded datasets without having to combine watermarks during watermark detection. Table 21 provides an example security report against mean collusion attacks using different configurations. In the first configuration two colluders (watermark number 1 and 2) attacked the watermarked fragment, in the second configuration three colluders (watermark number 1, 2 and 3) attacked the watermarked fragment, and so forth. The matching watermark similarity of each colluder is markedly higher compared to the watermark similarities of other watermarks.

Fragment 2017-02-04, Secret Key = 1, Watermark Number = 1 … 5, Maximum Error = 0.5, Number of Ranges = 10						
Colluders	Matching Fragment Similarity	Matching Watermark Similarity				
		1	2	3	4	5
1, 2	99.90 %	**94.32 %**	**94.32 %**	89.21 %	90.89 %	90.27 %
1, 2, 3	99.91 %	**93.55 %**	**92.92 %**	**92.81 %**	90.80 %	90.82 %
1, 2, 3, 4	99.92 %	**93.11 %**	**92.86 %**	**92.23 %**	**93.10 %**	90.85 %
1, 2, 3, 4, 5	99.93 %	**93.13 %**	**92.62 %**	**92.00 %**	**92.63 %**	**92.68 %**

Table 21: Example Security against Mean Collusion Attacks

Summarizing WP-7, watermark similarities of embedded watermarks were markedly higher than watermark similarities of other watermarks in every single attack scenario. The prototype may be considered secure against rounding and random value attacks at least by the watermark combination of multiple fragments if the attacks do not modify the data exceeding the usability constraint. Because of the time dependent matching measurements analysis, the prototype may be also be considered secure against deletion and subset selection attacks. Nevertheless, the time dependent detection also results in a vulnerability against attacks targeting the time field, e.g. time shifting attacks. Finally, the prototype can be considered secure against mean collusion attacks especially if watermark combinations are computed.

WP-8: Cipher keys shall not be used, but watermark keys shall be used. Cipher keys are not used if no encryption is performed, and watermark keys are used if watermarking is secured by secret watermark keys.
Cipher keys are not used, because no information is encrypted in watermarks. Watermark keys are used in watermark generation to secure the algorithm by seeding the PRNG for selecting the errors.
This requirement is fulfilled because no encryption is performed, and watermarking is secured by secret watermark keys.

WP-9: Modification and multiple watermarks shall not be used. Modification are not used if watermarks are not modified, and multiple watermarks are not used if fragments are watermarked only once per request.
Neither modifications nor multiple watermarks are even considered in the design and implementation of watermark embedding and detection, because they are not needed in this application of data leakage detection.
This requirement is fulfilled because watermarks are not modified, and fragments are only watermarked once per request.

WP-10: The prototype should be preferably cost-efficient. The cost-efficiency is measured by the sub-design goals. The cost-efficiency studies were conducted on a Lenovo Thinkpad T470p with a locally installed PostgreSQL[5] database and the file system utilized for data exchange. It should be noted that the time totals in the Tables 22-24 are more than the sum of the listed components because they include computational overhead.

Number of Fragments	Test	Fragmentation	Database Insert	Total
1	1	49 ms	239 ms	293 ms
	2	44 ms	223 ms	272 ms
	3	46 ms	243 ms	294 ms
	Average	*46 ms*	*235 ms*	*286 ms*
7	1	139 ms	671 ms	815 ms
	2	133 ms	642 ms	780 ms
	3	129 ms	762 ms	895 ms
	Average	*134 ms*	*692 ms*	*830 ms*
29	1	226 ms	1 975 ms	2 207 ms
	2	218 ms	1 992 ms	2 214 ms
	3	216 ms	2 012 ms	2 233 ms
	Average	*220 ms*	*1 993 ms*	*2 218 ms*

Table 22: Example Fragmentation Performance

WP-10a: The fragmentation should be preferably fast. The fragmentation performance is measured by the time required for fragmentation of differently sized datasets.
The fragmentation performance depends on the number of measurements of a to-be stored dataset. Table 22 provides an example performance report of storing differently sized datasets.
The fragmentation performance can be considered sufficient because the additional time required for fragmentation is negligible compared to the database inserts.

WP-10b: The watermark embedding should be preferably fast. The watermark embedding performance is measured by the time required for watermark embedding of differently sized datasets.

[5] PostgreSQL: https://www.postgresql.org/

The watermark embedding performance depends on the number of requested fragments. Table 23 provides an example performance report of watermark embedding of differently sized requested datasets. The report shows that the time required for watermark embedding increases almost linearly for every additional requested fragment.
The watermark embedding performance may be considered sufficient for practical use because the time required for watermarking a dataset with 29 fragments is on average just 5.6 seconds.

Number of Fragments	Test	Database Retrieval	Watermark Embedding	Dataset Providing	Total
1	1	199 ms	219 ms	97 ms	524 ms
	2	218 ms	215 ms	103 ms	545 ms
	3	198 ms	235 ms	127 ms	568 ms
	Average	*205 ms*	*223 ms*	*109 ms*	*546 ms*
7	1	314 ms	1 313 ms	587 ms	2 224 ms
	2	335 ms	1 304 ms	562 ms	2 211 ms
	3	339 ms	1 301 ms	569 ms	2 219 ms
	Average	*329 ms*	*1 306 ms*	*573 ms*	*2 218 ms*
29	1	416 ms	5 577 ms	2 331 ms	8 333 ms
	2	391 ms	5 746 ms	2 336 ms	8 483 ms
	3	422 ms	5 511 ms	2 342 ms	8 284 ms
	Average	*410 ms*	*5 611 ms*	*2 336 ms*	*8 367 ms*

Table 23: Example Watermark Embedding Performance

WP-10c: The watermark detection should be as fast as necessary. The watermark detection performance is measured by the time required for watermark detection of a suspicious fragment using different database contents.
The watermark detection performance depends on the number of relevant fragments used in the fragment similarity search, the number of watermarks of the matching fragment and the number of colluders given to the detection. In this evaluation, the performance impact of the number of colluders is not evaluated, because computing all possible combinations of multiple watermarks quickly escalates.
Table 24 provides an example performance report of watermark detection using different database contents. For this evaluation, we pseudo randomly generated fragments with equal timestamps such that fragment similarity search has to be performed for every relevant fragment. Furthermore, we set up a different number of watermarks for the matching fragment such that watermark similarity search has to be performed for each watermark. The report shows that for every additional fragment and every additional watermark, the detection performance decreases.
The watermark detection performance may be considered sufficient for practical use because the time required for watermark detection is not subject of high-performance demands like watermark embedding.

Number of Fragments	Number of Watermarks	Fragmentation	Watermark Detection	Total
1	1	50 ms	240 ms	321 ms
1	100	42 ms	559 ms	606 ms
100	1	47 ms	1 663 ms	1 716 ms
100	100	43 ms	1 987 ms	2 035 ms
1 000	1	46 ms	24 690 ms	24 741 ms
1 000	100	40 ms	25 025 ms	25 071 ms

Table 24: Example Watermark Detection Performance

In summary, the performance of fragmentation, watermark embedding and detection may be considered sufficient for practical usage. Especially assuming a doctor who typically requests data of a day, week or month, the additional effort for watermark embedding is manageable.

In this chapter, the requirements and design goals were evaluated using the proof-of-concept prototype. Table 25 summarizes the evaluation and additionally provides the results for a better overview.

Requirements based on the Problem Statement		
PS-1	The prototype shall be based on watermarking.	Yes
PS-2	The prototype shall be applicable to medical sensor data.	Yes
PS-3	The prototype shall detect data leakage.	Yes
Requirements based on the Solution Idea		
SI-1	The prototype shall implement the use of dataset fragments.	Yes
SI-2	The prototype shall implement the proposed watermark embedding.	
SI-2a	The watermark generation shall be configurable by a requested fragment.	Yes
SI-2b	The watermark generation shall be configurable by a usability constraint.	Yes
SI-2c	The watermark generation shall be configurable by a watermark number.	Yes
SI-2d	The watermark generation shall generate a matching watermark.	Yes
SI-2e	The generated watermark shall be embedded.	Yes
SI-3	The prototype shall implement the proposed watermark detection.	
SI-3a	The fragment similarity search shall identify a matching fragment.	Yes
SI-3b	The embedded watermark shall be extracted.	Yes
SI-3c	The already embedded watermarks shall be re-generated.	Yes
SI-3d	The watermark similarity search shall identify matching watermarks.	Yes
SI-3e	The leaker identification shall identify potential data leakers.	Yes
Requirements and Design Goals based on Watermarking Properties		
WP-1	The embedding effectiveness shall be 100%.	Yes
WP-2	The fidelity should be preferably high.	Considered high.
WP-3	The data payload should be as high as necessary.	Considered as high as necessary.
WP-4	The detection shall be informed.	Yes
WP-5	The false positive rate should be preferably low.	Considered very low.
WP-6	The robustness should be preferably high.	WP-7
WP-7	The security should be preferably high.	
WP-7a	The prototype should be secure against rounding value attacks.	Considered secure.
WP-7b	The prototype should be secure against random value attacks.	Considered secure.
WP-7c	The prototype should be secure against deletion attacks.	Considered secure.
WP-7d	The prototype should be secure against subset selection attacks.	Considered secure.
WP-7e	The prototype should be secure against mean collusion attacks.	Considered secure.
WP-8	Cipher keys shall not be used, but watermark keys shall be used.	Yes
WP-9	Modification and multiple watermarks shall not be used.	Yes
WP-10	The prototype should be preferably cost-efficient.	
WP-10a	The fragmentation should be preferably fast.	Considered sufficient.
WP-10b	The watermark embedding should be preferably fast.	Considered sufficient.
WP-10c	The watermark detection should be as fast as necessary.	Considered sufficient.

Table 25: Summary of the Prototype Evaluation

7. Conclusion

In this thesis, a new watermarking approach of medical sensor data for data leakage detection has been introduced. The watermarking approach is informed meaning that watermark embedding and detection take advantage of an existing copy of the original data. The watermark embedding utilizes the original data to generate and embed matching watermarks improving imperceptibility. The watermark detection utilizes the original data to extract the watermark and to identify potential data leakers. In addition, watermark embedding is configurable by usability constraints which depend on the sensor type of the measurements being watermarked. More precisely, a usability constraint limits the errors generated by watermark generation. The tighter the usability constraint, the more imperceptible but less secure the watermark is. The determination of an imperceptible but secure usability constraint is one of the major challenges for a data provider using this approach.

In this thesis, a proof-of-concept prototype based on the introduced watermarking approach has been designed, implemented, and evaluated. We have provided an overview of the state of the art of digital watermarking. Furthermore, requirements and design goals for the prototype are determined with respect to the problem statement of this thesis, the introduced watermarking approach and watermarking properties. The prototype is designed according to the watermarking approach, and expected results and limitations indicating future work are discussed. The prototype is implemented and evaluated regarding its requirements and design goals. It is shown, among others, that the prototype can be considered secure against common variants of distortion, collusion, subset selection and deletion attacks. The prototype, however, is vulnerable to attacks targeting the time field, e.g. time shifting attacks, due to time dependent watermark detection. The evaluation also showed that the performance of watermark embedding and detection can be considered sufficient for practical use of providing continuous blood glucose measurements for data users such as doctors.

The prototype is designed especially for medical sensor data but may also be applied to different kinds of sensor data by using different usability constraints. Furthermore, the prototype provides high extensibility and adaptability, because the algorithms of watermark embedding and detection can be arbitrary extended or adapted. In addition, watermark detection can be implemented or further improved even if a version of the prototype is already in production. The detection, for example, may be improved by adding security against new attacking methods or attack combinations.

Future work can be conducted regarding improving watermark embedding and detection. Regarding watermark embedding, especially watermark generation may be improved to enable fast generation of maximally different and very well matching watermarks. Regarding watermark detection, improvement can be made by more time independent similarity searches, criteria to limit the relevant fragments for fragment similarity search, and considering multiple matching fragments.

8. References

[1] X. Yi, A. Bouguettaya, D. Georgakopoulos, A. Song, and J. Willemson, "Privacy Protection for Wireless Medical Sensor Data," *IEEE Transactions on Dependable and Secure Computing*, vol. 13, no. 3, pp. 369–380, 2016, doi: 10.1109/TDSC.2015.2406699.

[2] P. Papadimitriou and H. Garcia-Molina, "Data Leakage Detection," *IEEE Transactions on Knowledge and Data Engineering*, vol. 23, no. 1, pp. 51–63, 2011, doi: 10.1109/TKDE.2010.100.

[3] I. Cox, M. Miller, J. Bloom, J. Fridrich, and T. Kalker, *Digital Watermarking and Steganography*: Morgan Kaufmann Publishers Inc., 2007.

[4] T. V. Bui, B. Q. Nguyen, T. D. Nguyen, N. Sonehara, and I. Echizen, "Robust Fingerprinting Codes for Database," in *Lecture Notes in Computer Science, Algorithms and Architectures for Parallel Processing*, D. Hutchison et al., Eds., Cham: Springer International Publishing, 2013, pp. 167–176.

[5] R. Halder, S. Pal, and A. Cortesi, "Watermarking Techniques for Relational Databases: Survey, Classification and Comparison," *Journal of Universal Computer Science*, no. 16, 2010.

[6] T. Battelino *et al.,* "Clinical Targets for Continuous Glucose Monitoring Data Interpretation: Recommendations From the International Consensus on Time in Range," *Diabetes care*, vol. 42, no. 8, pp. 1593–1603, 2019, doi: 10.2337/dci19-0028.

[7] S. Saini, "A survey on watermarking web contents for protecting copyright," in *2015 International Conference on Innovations in Information, Embedded and Communication Systems (ICIIECS)*, Coimbatore, India, Mar. 2015 - Mar. 2015, pp. 1–4.

[8] S. A. Craver, M. Wu, and B. Liu, "What can we reasonably expect from watermarks?," in *Proceedings of the 2001 IEEE Workshop on the Applications of Signal Processing to Audio and Acoustics (Cat. No.01TH8575)*, New Platz, NY, USA, Oct. 2001, pp. 223–226.

[9] M. Pal, "A Survey on Digital Watermarking and its Application," *International Journal of Advanced Computer Science and Applications*, vol. 7, no. 1, pp. 153–156, 2016.

[10] 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, doi: 10.1007/s00778-003-0097-x.

[11] N. Agarwal, A. K. Singh, and P. K. Singh, "Survey of robust and imperceptible watermarking," *Multimed Tools Appl*, vol. 78, no. 7, pp. 8603–8633, 2019, doi: 10.1007/s11042-018-7128-5.

[12] W. A. W. Adnan, S. Hitam, S. Abdul-Karim, and M. R. Tamjis, "A review of image watermarking," in *Proceedings. Student Conference on Research and Development, 2003. SCORED 2003*, Putrajaya, Malaysia, Aug. 2003, pp. 381–384.

[13] I. J. Cox, M. L. Miller, and J. A. Bloom, "Watermarking applications and their properties," in *Proceedings International Conference on Information Technology: Coding and Computing (Cat. No.PR00540)*, Las Vegas, NV, USA, Mar. 2000, pp. 6–10.

[14] 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)*, Pusan, South Korea, Jun. 2001, pp. 272–277.

[15] U. H. Panchal and R. Srivastava, "A Comprehensive Survey on Digital Image Watermarking Techniques," in *2015 Fifth International Conference on Communication Systems and Network Technologies*, Gwalior, India, Apr. 2015 - Apr. 2015, pp. 591–595.

[16] Y. Uchida, Y. Nagai, S. Sakazawa, and S.'i. Satoh, "Embedding Watermarks into Deep Neural Networks," in *Proceedings of the 2017 ACM on International Conference on Multimedia Retrieval - ICMR '17*, Bucharest, Romania, 2017, pp. 269–277.

[17] A. Soltani Panah, R. van Schyndel, T. Sellis, and E. Bertino, “On the Properties of Non-Media Digital Watermarking: A Review of State of the Art Techniques,” *IEEE Access*, vol. 4, pp. 2670–2704, 2016, doi: 10.1109/ACCESS.2016.2570812.
[18] A. A. Mohanpurkar and M. S. Joshi, “Applying watermarking for copyright protection, traitor identification and joint ownership: A review,” in *2011 World Congress on Information and Communication Technologies*, Mumbai, India, Dec. 2011 - Dec. 2011, pp. 1014–1019.
[19] Y. Li, V. Swarup, and S. Jajodia, “Fingerprinting relational databases: schemes and specialties,” *IEEE Trans. Dependable and Secure Comput.*, vol. 2, no. 1, pp. 34–45, 2005, doi: 10.1109/TDSC.2005.12.
[20] V. Khanduja, “Database watermarking, a technological protective measure: Perspective, security analysis and future directions,” *Journal of Information Security and Applications*, vol. 37, pp. 38–49, 2017, doi: 10.1016/j.jisa.2017.10.001.
[21] M.-R. Xie, C.-C. Wu, J.-J. Shen, and M.-S. Hwang, “A Survey of Data Distortion Watermarking Relational Databases,” *IJ Network Security*, vol. 18, no. 6, pp. 1022–1033, 2016.
[22] M. Kamran and M. Farooq, “A Comprehensive Survey of Watermarking Relational Databases Research,” Jan. 2018. [Online]. Available: http://arxiv.org/pdf/1801.08271v1
[23] S. Liu, S. Wang, R. H. Deng, and W. Shao, “A Block Oriented Fingerprinting Scheme in Relational Database,” in *Information Security and Cryptology - ICISC 2004*, 2005, pp. 455–466.
[24] N. R. Wagner, “Fingerprinting,” in *1983 IEEE Symposium on Security and Privacy*, Oakland, CA, USA, Apr. 1983 - Apr. 1983, p. 18.
[25] T. P. Duy, D. Tran, and W. Ma, “An intelligent learning-based watermarking scheme for outsourced biomedical time series data,” in *2017 International Joint Conference on Neural Networks (IJCNN)*, Anchorage, AK, USA, May. 2017 - May. 2017, pp. 4408–4415.
[26] T. P. Duy, D. Tran, and W. Ma, “Ownership protection of outsourced biomedical time seried data based on optimal watermarking scheme in data mining,”
[27] E. Ayday, E. Yilmaz, and A. Yilmaz, “Robust Optimiation-Based Watermarking Scheme for Sequential Data,” in *22nd International Symposium on Research in Attacks, Intrusions and Defenses (RAID 2019)*, 2019, pp. 323–336. [Online]. Available: https://www.usenix.org/conference/raid2019/presentation/ayday
[28] R. M. Bergenstal *et al.,* “Glucose Management Indicator (GMI): A New Term for Estimating A1C From Continuous Glucose Monitoring,” *Diabetes care*, vol. 41, no. 11, pp. 2275–2280, 2018, doi: 10.2337/dc18-1581.
[29] S. Suh and J. H. Kim, “Glycemic Variability: How Do We Measure It and Why Is It Important?,” *Diabetes & metabolism journal*, vol. 39, no. 4, pp. 273–282, 2015, doi: 10.4093/dmj.2015.39.4.273.

List of Figures

List of Tables

List of Equations

Appendix A Installation Guide

The prerequisite for executing the prototype is the database setup. Either the following database setup is used or the connections in the DatabaseService class must be adapted. The PostgreSQL[6] database needs to be installed locally and be using the port 5432. Also, there must be a database called "watermarking". The login data must be the user "postgres" with the password "admin". In addition, the tables must be created according to the following SQL script. For the sake of simplicity, we refrain from using foreign keys or additional tables for e.g. data user management.

```
CREATE TABLE fragment (
        device_id text,
        measurements jsonb,
        type text,
        date date,
        unit text,
        secret_key bigint,
        CONSTRAINT fragment_pkey PRIMARY KEY (device_id, type, date, unit)
)
CREATE TABLE request (
        device_id text,
        type text,
        unit text,
        date date,
        timestamps timestamp without time zone[],
        number_of_watermark integer,
        data_user integer,
        CONSTRAINT request_pkey PRIMARY KEY (data_user, date, unit, type, device_id)
)
CREATE TABLE usability_constraint (
        type text,
        unit text,
        maximum_error numeric,
        minimum_value numeric,
        maximum_value numeric,
        number_of_ranges integer,
        CONSTRAINT usability_constraint_pkey PRIMARY KEY (type, unit)
)
```

After the database is set up, the prototype can be executed as Java program or as jar file. As java program, the ProgramUI class is used to execute simulators and the "files" folder in the project is used for the data. The prototype can be packaged into a jar file by running the maven install command. The resulting jar file is put into the "C:/temp" directory. In this case, the folder used for the data is the "C:/temp/files" folder. Also make sure that the "C:/temp/files" folder contains the files with the test data. The jar file can be executed in the command line or in batch files by *java -jar C:/temp/prototype-1.jar*.

[6] PostgreSQL: https://www.postgresql.org/

Appendix B Source Code

watermarking/pom.xml

```xml
<project xmlns="http://maven.apache.org/POM/4.0.0"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/maven-4.0.0.xsd">
   <modelVersion>4.0.0</modelVersion>
   <groupId>watermarking</groupId>
   <artifactId>prototype</artifactId>
   <version>1</version>
   <packaging>jar</packaging>

   <build>
      <plugins>
         <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-shade-plugin</artifactId>
            <version>3.2.1</version>
            <executions>
               <execution>
                  <phase>package</phase>
                  <goals>
                     <goal>shade</goal>
                  </goals>
                  <configuration>
                     <transformers>
                        <transformer

   implementation="org.apache.maven.plugins.shade.resource.ManifestResourceTransformer">
                           <mainClass>ui.ConsoleUI</mainClass>
                        </transformer>
                     </transformers>
                  </configuration>
               </execution>
            </executions>
         </plugin>
         <plugin>
            <groupId>org.apache.maven.plugins</groupId>
            <artifactId>maven-jar-plugin</artifactId>
            <version>2.3.1</version>
            <configuration>
               <outputDirectory>C:\temp</outputDirectory>
            </configuration>
         </plugin>
      </plugins>
   </build>

   <dependencies>
      <!-- https://mvnrepository.com/artifact/postgresql/postgresql -->
      <dependency>
         <groupId>postgresql</groupId>
         <artifactId>postgresql</artifactId>
         <version>9.1-901.jdbc4</version>
      </dependency>
      <!-- https://mvnrepository.com/artifact/com.googlecode.json-simple/json-simple -->
      <dependency>
         <groupId>com.googlecode.json-simple</groupId>
         <artifactId>json-simple</artifactId>
         <version>1.1.1</version>
      </dependency>
      <!-- https://mvnrepository.com/artifact/org.apache.commons/commons-math3 -->
      <dependency>
         <groupId>org.apache.commons</groupId>
         <artifactId>commons-math3</artifactId>
         <version>3.6.1</version>
      </dependency>
   </dependencies>

</project>
```

watermarking/src/main/java/entities/DataUserWatermark.java

```
package entities;

import java.math.BigDecimal;
import java.util.List;

public class DataUserWatermark implements Comparable<DataUserWatermark> {

   private BigDecimal probability;
   private List<Integer> dataUsers;
   private BigDecimal[] watermark;

   public DataUserWatermark(BigDecimal probability, List<Integer> dataUsers) {
      this.probability = probability;
      this.dataUsers = dataUsers;
   }
   public DataUserWatermark(List<Integer> dataUsers, BigDecimal[] watermark) {
      this.dataUsers = dataUsers;
      this.watermark = watermark;
   }

   @Override
   public String toString() {
      return "probability of " + probability + "\tby data user " + dataUsers.toString();
   }

   @Override
   public int compareTo(DataUserWatermark o) {
      if (getProbability() == null || o.getProbability() == null) {
         return 0;
      }
      return o.getProbability().compareTo(getProbability());
   }

   @Override
   public boolean equals(Object obj) {
      if (this == obj)
         return true;
      if (obj == null)
         return false;
      if (getClass() != obj.getClass())
         return false;
      DataUserWatermark other = (DataUserWatermark) obj;
      if (dataUsers == null) {
         if (other.dataUsers != null)
            return false;
      } else if (!dataUsers.equals(other.dataUsers))
         return false;
      return true;
   }

   public BigDecimal getProbability() {
      return probability;
   }
   public void setProbability(BigDecimal probability) {
      this.probability = probability;
   }
   public List<Integer> getDataUsers() {
      return dataUsers;
   }
   public void setDataUsers(List<Integer> dataUsers) {
      this.dataUsers = dataUsers;
   }
   public BigDecimal[] getWatermark() {
      return watermark;
   }
   public void setWatermark(BigDecimal[] watermark) {
      this.watermark = watermark;
   }
}
```

watermarking/src/main/java/entities/Fragment.java

```java
package entities;

import java.math.BigDecimal;
import java.time.LocalDate;
import java.time.LocalDateTime;
import java.time.format.DateTimeFormatter;
import java.util.LinkedList;
import java.util.List;

import org.json.simple.JSONArray;
import org.json.simple.JSONObject;
import org.json.simple.parser.JSONParser;
import org.json.simple.parser.ParseException;

public class Fragment implements Comparable<Fragment> {

   private String deviceId;
   private String type;
   private String unit;
   private LocalDate date;
   private long secretKey;
   private List<Measurement> measurements;

   public Fragment() {}
   public Fragment(String deviceId, String type, String unit, LocalDate date) {
      this.deviceId = deviceId;
      this.type = type;
      this.unit = unit;
      this.date = date;
      this.measurements = new LinkedList<Measurement>();
   }
   public Fragment(String deviceId, String type, String unit, LocalDate date, long secretKey) {
      this.deviceId = deviceId;
      this.type = type;
      this.unit = unit;
      this.date = date;
      this.secretKey = secretKey;
      this.measurements = new LinkedList<Measurement>();
   }

   public void setMeasurementsFromJsonArrayString(String measurements) {
      this.measurements = new LinkedList<Measurement>();

      try {
         DateTimeFormatter formatter = DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm:ss");
         JSONParser parser = new JSONParser();
         JSONArray array = (JSONArray) parser.parse(measurements);

         for (Object object : array) {
            JSONObject measurementObject = (JSONObject) object;

            String deviceId = (String) measurementObject.get("deviceId");
            String type = (String) measurementObject.get("type");
            String unit = (String) measurementObject.get("unit");
            String timeString = (String) measurementObject.get("time");

            if (timeString.contains("T")) {
               timeString = timeString.replace("T", " ");
            }

            LocalDateTime time = LocalDateTime.parse(timeString, formatter);
            BigDecimal value = new BigDecimal(measurementObject.get("value").toString());

            this.measurements.add(new Measurement(deviceId, type, unit, time, value));
         }
      } catch (ParseException ex) {
         ex.printStackTrace();
      }
   }

   public String getMeasurementsAsJsonArrayString() {
      return "[" + getMeasurementsAsJsonString() + "\n]";
   }

   public String getMeasurementsAsJsonString() {
      String json = new String();
```

```
        for (int i = 0; i < getMeasurements().size(); i++) {
            Measurement measurement = getMeasurements().get(i);
            json = json + "\n\t{";
            json = json + "\n\t\t\"deviceId\": \"" + measurement.getDeviceId() + "\",";
            json = json + "\n\t\t\"type\": \"" + measurement.getType() + "\",";
            json = json + "\n\t\t\"unit\": \"" + measurement.getUnit() + "\",";
            json = json + "\n\t\t\"time\": \"" +
                measurement.getTime().format(DateTimeFormatter.ISO_LOCAL_DATE_TIME) + "\",";
            json = json + "\n\t\t\"value\": " + measurement.getValue();
            if (i + 1 < getMeasurements().size())
                json = json + "\n\t},";
            else {
                json = json + "\n\t}";
            }
        }
        return json;
    }

    @Override
    public int compareTo(Fragment o) {
        if (getDate() == null || o.getDate() == null) {
            return 0;
        }
        return getDate().compareTo(o.getDate());
    }

    @Override
    public boolean equals(Object obj) {
        if (this == obj)
            return true;
        if (obj == null)
            return false;
        if (getClass() != obj.getClass())
            return false;
        Fragment other = (Fragment) obj;
        if (date == null) {
            if (other.date != null)
                return false;
        } else if (!date.equals(other.date))
            return false;
        if (deviceId == null) {
            if (other.deviceId != null)
                return false;
        } else if (!deviceId.equals(other.deviceId))
            return false;
        if (type == null) {
            if (other.type != null)
                return false;
        } else if (!type.equals(other.type))
            return false;
        if (unit == null) {
            if (other.unit != null)
                return false;
        } else if (!unit.equals(other.unit))
            return false;
        return true;
    }

    public String getDeviceId() {
        return deviceId;
    }
    public void setDeviceId(String deviceId) {
        this.deviceId = deviceId;
    }
    public String getType() {
        return type;
    }
    public void setType(String type) {
        this.type = type;
    }
    public String getUnit() {
        return unit;
    }
    public void setUnit(String unit) {
        this.unit = unit;
    }
    public LocalDate getDate() {
        return date;
    }
    public void setDate(LocalDate date) {
        this.date = date;
    }
    public long getSecretKey() {
        return secretKey;
    }
    public void setSecretKey(long secretKey) {
        this.secretKey = secretKey;
    }
    public List<Measurement> getMeasurements() {
        return measurements;
    }
    public void setMeasurements(List<Measurement> measurements) {
        this.measurements = measurements;
    }
}
```

watermarking/src/main/java/entities/Measurement.java

```
package entities;

import java.math.BigDecimal;
import java.time.LocalDateTime;

public class Measurement implements Comparable<Measurement> {

   private String deviceId;
   private String type;
   private String unit;
   private LocalDateTime time;
   private BigDecimal value;

   public Measurement() {}
   public Measurement(String deviceId, String type, String unit, LocalDateTime time, BigDecimal value) {
      this.deviceId = deviceId;
      this.type = type;
      this.unit = unit;
      this.time = time;
      this.value = value;
   }

   @Override
   public int compareTo(Measurement o) {
      if (getTime() == null || o.getTime() == null) {
         return 0;
      }
      return getTime().compareTo(o.getTime());
   }

   public String getDeviceId() {
      return deviceId;
   }
   public void setDeviceId(String deviceId) {
      this.deviceId = deviceId;
   }
   public String getType() {
      return type;
   }
   public void setType(String type) {
      this.type = type;
   }
   public String getUnit() {
      return unit;
   }
   public void setUnit(String unit) {
      this.unit = unit;
   }
   public LocalDateTime getTime() {
      return time;
   }
   public void setTime(LocalDateTime time) {
      this.time = time;
   }
   public BigDecimal getValue() {
      return value;
   }
   public void setValue(BigDecimal value) {
      this.value = value;
   }
}
```

watermarking/src/main/java/entities/Range.java

```java
package entities;

import java.math.BigDecimal;

public class Range<T> implements Comparable<Range<T>> {

   private BigDecimal minimum;
   private BigDecimal maximum;
   private T value;

   public Range(BigDecimal minimum, BigDecimal maximum) {
      this.minimum = minimum;
      this.maximum = maximum;
   }
   public Range(BigDecimal minimum, BigDecimal maximum, T value) {
      this.minimum = minimum;
      this.maximum = maximum;
      this.value = value;
   }

   @Override
   public int compareTo(Range<T> o) {
      if (getMinimum() == null || o.getMinimum() == null) {
         return 0;
      }
      return getMinimum().compareTo(o.getMinimum());
   }

   @Override
   public boolean equals(Object obj) {
      if (this == obj)
         return true;
      if (obj == null)
         return false;
      if (getClass() != obj.getClass())
         return false;
      Range<?> other = (Range<?>) obj;
      if (maximum == null) {
         if (other.maximum != null)
            return false;
      } else if (!maximum.equals(other.maximum))
         return false;
      if (minimum == null) {
         if (other.minimum != null)
            return false;
      } else if (!minimum.equals(other.minimum))
         return false;
      return true;
   }

   public BigDecimal getMinimum() {
      return minimum;
   }
   public void setMinimum(BigDecimal minimum) {
      this.minimum = minimum;
   }
   public BigDecimal getMaximum() {
      return maximum;
   }
   public void setMaximum(BigDecimal maximum) {
      this.maximum = maximum;
   }
   public T getValue() {
      return value;
   }
   public void setValue(T value) {
      this.value = value;
   }
}
```

watermarking/src/main/java/entities/Request.java

```
package entities;

import java.time.LocalDate;
import java.time.LocalDateTime;
import java.util.ArrayList;

public class Request {

   private String deviceId;
   private int dataUser;
   private String type;
   private String unit;
   private LocalDate date;
   private int numberOfWatermark;
   private ArrayList<LocalDateTime> timestamps;

   public Request(String deviceId, int dataUser, String type, String unit, LocalDate date, int
         numberOfWatermark, ArrayList<LocalDateTime> timestamps) {
      this.deviceId = deviceId;
      this.dataUser = dataUser;
      this.type = type;
      this.unit = unit;
      this.date = date;
      this.numberOfWatermark = numberOfWatermark;
      this.timestamps = timestamps;
   }

   public String getDeviceId() {
      return deviceId;
   }
   public void setDeviceId(String deviceId) {
      this.deviceId = deviceId;
   }
   public int getDataUser() {
      return dataUser;
   }
   public void setDataUser(int dataUser) {
      this.dataUser = dataUser;
   }
   public String getType() {
      return type;
   }
   public void setType(String type) {
      this.type = type;
   }
   public String getUnit() {
      return unit;
   }
   public void setUnit(String unit) {
      this.unit = unit;
   }
   public LocalDate getDate() {
      return date;
   }
   public void setDate(LocalDate date) {
      this.date = date;
   }
   public int getNumberOfWatermark() {
      return numberOfWatermark;
   }
   public void setNumberOfWatermark(int numberOfWatermark) {
      this.numberOfWatermark = numberOfWatermark;
   }
   public ArrayList<LocalDateTime> getTimestamps() {
      return timestamps;
   }
   public void setTimestamps(ArrayList<LocalDateTime> timestamps) {
      this.timestamps = timestamps;
   }
}
```

watermarking/src/main/java/entities/UsabilityConstraint.java

```
package entities;

import java.math.BigDecimal;

public class UsabilityConstraint {

   private String type;
   private String unit;
   private Integer numberOfRanges;
   private BigDecimal minimumValue;
   private BigDecimal maximumValue;
   private BigDecimal maximumError;

   public UsabilityConstraint(Double maximumError, Integer numberOfRanges) {
      this.type = "cbg";
      this.unit = "mmol/L";
      this.minimumValue = BigDecimal.valueOf(0.0);
      this.maximumValue = BigDecimal.valueOf(55.0);
      this.maximumError = BigDecimal.valueOf(maximumError);
      this.numberOfRanges = numberOfRanges;
   }
   public UsabilityConstraint(String type, String unit, BigDecimal minimumValue, BigDecimal maximumValue,
         BigDecimal maximumError, Integer numberOfRanges) {
      this.type = type;
      this.unit = unit;
      this.minimumValue = minimumValue;
      this.maximumValue = maximumValue;
      this.maximumError = maximumError;
      this.numberOfRanges = numberOfRanges;
   }

   public String getType() {
      return type;
   }
   public void setType(String type) {
      this.type = type;
   }
   public String getUnit() {
      return unit;
   }
   public void setUnit(String unit) {
      this.unit = unit;
   }
   public BigDecimal getMinimumValue() {
      return minimumValue;
   }
   public void setMinimumValue(BigDecimal minimumValue) {
      this.minimumValue = minimumValue;
   }
   public BigDecimal getMaximumValue() {
      return maximumValue;
   }
   public void setMaximumValue(BigDecimal maximumValue) {
      this.maximumValue = maximumValue;
   }
   public BigDecimal getMaximumError() {
      return maximumError;
   }
   public void setMaximumError(BigDecimal maximumError) {
      this.maximumError = maximumError;
   }
   public Integer getNumberOfRanges() {
      return numberOfRanges;
   }
   public void setNumberOfRanges(Integer numberOfRanges) {
      this.numberOfRanges = numberOfRanges;
   }
}
```

watermarking/src/main/java/services/ContainerService.java

```java
package services;

import java.util.List;
import java.util.Random;

import entities.Fragment;
import utilities.DatabaseService;
import utilities.FragmentationService;
import utilities.LogService;
import utilities.TimeService;

public class ContainerService {

   public static void storeDataset(Boolean randomSecretKey, String datasetName) {
      LogService.log(LogService.SERVICE_LEVEL, "ContainerService", "FragmentationService.getFragments");
      TimeService timeService = new TimeService();
      List<Fragment> fragments = FragmentationService.getFragments(datasetName);
      timeService.stop();
      LogService.log(LogService.SERVICE_LEVEL, "ContainerService", "FragmentationService.getFragments",
            timeService.getTime());

      LogService.log(LogService.SERVICE_LEVEL, "ContainerService", "generateSecretKeys");
      timeService = new TimeService();
      if (randomSecretKey == true) {
         Random random = new Random();
         for (Fragment fragment : fragments) {
            fragment.setSecretKey(random.nextLong());
         }
      } else {
         for (Fragment fragment : fragments) {
            fragment.setSecretKey(1);
         }
      }
      timeService.stop();
      LogService.log(LogService.SERVICE_LEVEL, "ContainerService", "generateSecretKeys",
            timeService.getTime());

      LogService.log(LogService.SERVICE_LEVEL, "ContainerService", "DatabaseService.insertFragments");
      timeService = new TimeService();
      DatabaseService.insertFragments(fragments);
      timeService.stop();
      LogService.log(LogService.SERVICE_LEVEL, "ContainerService", "DatabaseService.insertFragments",
            timeService.getTime());
   }

}
```

watermarking/src/main/java/services/DataService.java

```java
package services;

import java.math.BigDecimal;
import java.time.LocalDate;
import java.time.LocalDateTime;
import java.util.ArrayList;
import java.util.List;

import entities.Fragment;
import entities.Request;
import entities.UsabilityConstraint;
import utilities.DatabaseService;
import utilities.FileService;
import utilities.LogService;
import utilities.TimeService;
import utilities.WatermarkService;

public class DataService {

   public static void requestDataset(String datasetName, int dataUser, String deviceId, String type, String
         unit, LocalDate from, LocalDate to) {
      LogService.log(LogService.SERVICE_LEVEL, "DataService", "DatabaseService.getFragments");
      TimeService timeService = new TimeService();
      List<Fragment> fragments = DatabaseService.getFragments(deviceId, type, unit, from, to);
      timeService.stop();
      LogService.log(LogService.SERVICE_LEVEL, "DataService", "DatabaseService.getFragments",
         timeService.getTime());

      LogService.log(LogService.SERVICE_LEVEL, "DataService", "watermarkEmbedding");
      timeService = new TimeService();
      fragments = embedWatermarks(dataUser, fragments);
      timeService.stop();
      LogService.log(LogService.SERVICE_LEVEL, "DataService", "watermarkEmbedding", timeService.getTime());

      LogService.log(LogService.SERVICE_LEVEL, "DataService", "FileService.writeDataset");
      timeService = new TimeService();
      FileService.writeDataset(datasetName, fragments);
      timeService.stop();
      LogService.log(LogService.SERVICE_LEVEL, "DataService", "FileService.writeDataset",
         timeService.getTime());
   }

   public static void requestDataset(String datasetName, int dataUser, int numberOfDevices, String type,
         String unit, LocalDate from, LocalDate to) {
      LogService.log(LogService.SERVICE_LEVEL, "DataService", "DatabaseService.getFragments");
      TimeService timeService = new TimeService();
      List<Fragment> fragments = DatabaseService.getFragments(numberOfDevices, type, unit, from, to);
      timeService.stop();
      LogService.log(LogService.SERVICE_LEVEL, "DataService", "DatabaseService.getFragments",
         timeService.getTime());

      LogService.log(LogService.SERVICE_LEVEL, "DataService", "watermarkEmbedding");
      timeService = new TimeService();
      fragments = embedWatermarks(dataUser, fragments);
      timeService.stop();
      LogService.log(LogService.SERVICE_LEVEL, "DataService", "watermarkEmbedding", timeService.getTime());

      LogService.log(LogService.SERVICE_LEVEL, "DataService", "FileService.writeDataset");
      timeService = new TimeService();
      FileService.writeDataset(datasetName, fragments);
      timeService.stop();
      LogService.log(LogService.SERVICE_LEVEL, "DataService", "FileService.writeDataset",
         timeService.getTime());
   }

   private static List<Fragment> embedWatermarks(int dataUser, List<Fragment> fragments) {
      LocalDateTime timestamp = LocalDateTime.now();

      for (int i = 0; i < fragments.size(); i++) {
         Fragment fragment = fragments.get(i);

         UsabilityConstraint usabilityConstraint = DatabaseService.getUsabilityConstraint(fragment.getType(),
               fragment.getUnit());
         Request request = DatabaseService.getRequest(dataUser, fragment.getDeviceId(), fragment.getType(),
               fragment.getUnit(), fragment.getDate());
```

```
            if (request == null) {
               int numberOfWatermark = 1 + DatabaseService.getNumberOfWatermark(fragment.getDeviceId(),
                     fragment.getType(), fragment.getUnit(), fragment.getDate());
               ArrayList<LocalDateTime> timestamps = new ArrayList<>();
               timestamps.add(timestamp);
               request = new Request(fragment.getDeviceId(), dataUser, fragment.getType(), fragment.getUnit(),
                     fragment.getDate(), numberOfWatermark, timestamps);
               DatabaseService.insertRequest(request);
            }
            else {
               request.getTimestamps().add(timestamp);
               DatabaseService.updateRequest(request);
            }

            BigDecimal[] watermark = WatermarkService.generateWatermark(request, usabilityConstraint, fragment);

            for (int j = 0; j < fragment.getMeasurements().size(); j++) {
               BigDecimal watermarkedValue = fragment.getMeasurements().get(j).getValue().add(watermark[j]);
               fragment.getMeasurements().get(j).setValue(watermarkedValue);
            }

            fragments.set(i, fragment);
         }
         return fragments;
      }
   }
```

watermarking/src/main/java/services/DetectionService.java

```
package services;

import java.math.BigDecimal;
import java.math.RoundingMode;
import java.util.Arrays;
import java.util.Collections;
import java.util.HashMap;
import java.util.Iterator;
import java.util.LinkedList;
import java.util.List;
import java.util.Map.Entry;

import org.apache.commons.math3.util.CombinatoricsUtils;

import entities.Fragment;
import entities.DataUserWatermark;
import entities.Measurement;
import entities.Request;
import entities.UsabilityConstraint;
import utilities.DatabaseService;
import utilities.FileService;
import utilities.FragmentationService;
import utilities.LogService;
import utilities.TimeService;
import utilities.WatermarkService;

public class DetectionService {

   public static void detectLeakage(String datasetName, String reportName, BigDecimal
         fragmentSimilarityThreshold, BigDecimal watermarkSimilarityThreshold, int numberOfColluders) {
      LogService.log(LogService.SERVICE_LEVEL, "DetectionService", "FragmentationService.getFragments");
      TimeService timeService = new TimeService();
      List<Fragment> suspiciousFragments = FragmentationService.getFragments(datasetName);
      timeService.stop();
      LogService.log(LogService.SERVICE_LEVEL, "DetectionService", "FragmentationService.getFragments",
            timeService.getTime());

      LogService.log(LogService.SERVICE_LEVEL, "DetectionService", "watermarkDetection");
      timeService = new TimeService();
      String report = detectWatermarks(suspiciousFragments, fragmentSimilarityThreshold,
         watermarkSimilarityThreshold, numberOfColluders);
      timeService.stop();
      LogService.log(LogService.SERVICE_LEVEL, "DetectionService", "watermarkDetection",
         timeService.getTime());

      LogService.log(LogService.SERVICE_LEVEL, "DetectionService", "FileService.writeFile");
      timeService = new TimeService();
      FileService.writeFile(reportName, report);
      timeService.stop();
      LogService.log(LogService.SERVICE_LEVEL, "DetectionService", "FileService.writeFile",
         timeService.getTime());
   }

   private static String detectWatermarks(List<Fragment> suspiciousFragments, BigDecimal
         fragmentSimilarityThreshold, BigDecimal watermarkSimilarityThreshold, int numberOfColluders) {

      List<List<DataUserWatermark>> setOfMatchingWatermarks = new LinkedList<>();

      String report = "watermark detection report";
      report = report + "\nfragment similarity threshold:\t" + fragmentSimilarityThreshold;
      report = report + "\nwatermark similarity threshold:\t" + watermarkSimilarityThreshold;
      report = report + "\nnumber of colluders:\t\t\t" + numberOfColluders;
      report = report + "\n";

      for (int i = 0; i < suspiciousFragments.size(); i++) {
         Fragment suspiciousFragment = suspiciousFragments.get(i);

         report = report + "\n[" + (i + 1) + "] fragment leakage";
         report = report + "\nsuspicious fragment:\t\t\t" + suspiciousFragment.getDeviceId();
         report = report + "\t" + suspiciousFragment.getType();
         report = report + "\t" + suspiciousFragment.getUnit();
         report = report + "\t" + suspiciousFragment.getDate();

         Fragment matchingFragment = new Fragment();
         BigDecimal matchingFragmentSimilarity = BigDecimal.valueOf(-1.0);
         HashMap<Integer, Integer> matchingMeasurements = new HashMap<>();
         UsabilityConstraint usabilityConstraint = DatabaseService
               .getUsabilityConstraint(suspiciousFragment.getType(), suspiciousFragment.getUnit());

         for (Fragment fragment : DatabaseService.getFragments(suspiciousFragment.getType(),
               suspiciousFragment.getUnit())) {
            HashMap<Integer, Integer> matches = getMatchingMeasurements(suspiciousFragment, fragment);

            BigDecimal fragmentSimilarity = getFragmentSimilarity(suspiciousFragment, fragment,
                  usabilityConstraint, matches);

            if (fragmentSimilarity.compareTo(matchingFragmentSimilarity) > 0) {
               matchingFragmentSimilarity = fragmentSimilarity;
               matchingFragment = fragment;
               matchingMeasurements = matches;
            }
         }

         if (matchingFragmentSimilarity.compareTo(fragmentSimilarityThreshold) >= 0) {
            report = report + "\nmatching original fragment:\t\t" + matchingFragment.getDeviceId();
            report = report + "\t" + matchingFragment.getType();
            report = report + "\t" + matchingFragment.getUnit();
            report = report + "\t" + matchingFragment.getDate();
            report = report + "\nmatching fragment similarity:\t" + matchingFragmentSimilarity;

            BigDecimal[] extractedWatermark = extractWatermark(suspiciousFragment, matchingFragment,
                  matchingMeasurements);

            List<Request> requests = DatabaseService.getRequests(matchingFragment.getDeviceId(),
                  matchingFragment.getType(), matchingFragment.getUnit(), matchingFragment.getDate());

            List<DataUserWatermark> matchingWatermarks = new LinkedList<>();

            for (DataUserWatermark dataUserWatermark : getSetOfWatermarks(requests, usabilityConstraint,
                  matchingFragment, numberOfColluders)) {
               BigDecimal watermarkSimilarity = getWatermarkSimilarity(extractedWatermark,
                     dataUserWatermark.getWatermark(), usabilityConstraint, matchingMeasurements);

               if (watermarkSimilarity.compareTo(watermarkSimilarityThreshold) >= 0) {
                  dataUserWatermark.setProbability(watermarkSimilarity);
                  matchingWatermarks.add(dataUserWatermark);
               }
            }

            if (matchingWatermarks.size() > 0) {
               Collections.sort(matchingWatermarks);
               report = report + "\nmatching watermarks: ";

               for (DataUserWatermark matchingWatermark : matchingWatermarks) {
                  report = report + "\n\t\t" + matchingWatermark.toString();
               }

               setOfMatchingWatermarks.add(matchingWatermarks);
            } else {
               report = report + "\nno matching watermarks detected";
            }

         } else {
            report = report + "\nno matching fragment detected";
         }
      }

      List<DataUserWatermark> datasetLeakers = getDatasetLeakers(suspiciousFragments.size(),
         setOfMatchingWatermarks);
      report = report + "\n\ndataset leakage";
      if (datasetLeakers.size() > 0) {
         Collections.sort(datasetLeakers);

         for (DataUserWatermark dataLeaker : datasetLeakers) {
            report = report + "\n\t\t" + dataLeaker;
         }
      } else {
         report = report + "\nno leakage detected";
      }

      return report;
   }
```

```
    private static HashMap<Integer, Integer> getMatchingMeasurements(Fragment suspiciousFragment,
            Fragment originalFragment) {
        HashMap<Integer, Integer> sequence = new HashMap<>();

        for (int i = 0; i < suspiciousFragment.getMeasurements().size(); i++) {
            Measurement suspiciousMeasurement = suspiciousFragment.getMeasurements().get(i);

            for (int j = 0; j < originalFragment.getMeasurements().size(); j++) {
                Measurement originalMeasurement = originalFragment.getMeasurements().get(j);

                if (originalMeasurement.getTime().isEqual(suspiciousMeasurement.getTime())) {
                    sequence.put(i, j);
                }
            }
        }
        return sequence;
    }

    private static BigDecimal getFragmentSimilarity(Fragment suspiciousFragment, Fragment originalFragment,
            UsabilityConstraint usabilityConstraint, HashMap<Integer, Integer> matchingMeasurements) {
        BigDecimal similarity = new BigDecimal("0.0");

        for (Entry<Integer, Integer> entry : matchingMeasurements.entrySet()) {
            Measurement suspiciousMeasurement = suspiciousFragment.getMeasurements().get(entry.getKey());
            Measurement originalMeasurement = originalFragment.getMeasurements().get(entry.getValue());

            BigDecimal distance =
                (suspiciousMeasurement.getValue().subtract(originalMeasurement.getValue())).abs();
            BigDecimal relativeDistance = distance.divide(usabilityConstraint.getMaximumValue(), 4,
                    RoundingMode.HALF_UP);

            BigDecimal measurementSimilarity = BigDecimal.valueOf(1).subtract(relativeDistance);
            similarity = similarity.add(measurementSimilarity);
        }

        similarity = similarity.divide(BigDecimal.valueOf(suspiciousFragment.getMeasurements().size()), 4,
                RoundingMode.HALF_UP);
        return similarity;
    }

    private static BigDecimal[] extractWatermark(Fragment suspiciousFragment, Fragment matchingFragment,
            HashMap<Integer, Integer> matchingMeasurements) {
        BigDecimal[] watermark = new BigDecimal[suspiciousFragment.getMeasurements().size()];

        for (Entry<Integer, Integer> entry : matchingMeasurements.entrySet()) {
            watermark[entry.getKey()] = suspiciousFragment.getMeasurements().get(entry.getKey()).getValue()
                    .subtract(matchingFragment.getMeasurements().get(entry.getValue()).getValue());
        }
        return watermark;
    }

    private static List<DataUserWatermark> getSetOfWatermarks(List<Request> requests,
            UsabilityConstraint usabilityConstraint, Fragment matchingFragment, int numberOfColluders) {
        List<DataUserWatermark> setOfWatermarks = new LinkedList<>();

        List<DataUserWatermark> singleWatermarks = new LinkedList<>();
        for (Request request : requests) {
            BigDecimal[] watermark = WatermarkService.generateWatermark(request, usabilityConstraint,
                matchingFragment);
            DataUserWatermark singleWatermark = new DataUserWatermark(Arrays.asList(request.getDataUser()),
                watermark);
            singleWatermarks.add(singleWatermark);
            setOfWatermarks.add(singleWatermark);
        }

        for (int i = 2; i < numberOfColluders + 1 && i <= singleWatermarks.size(); i++) {
            Iterator<int[]> iterator = CombinatoricsUtils.combinationsIterator(singleWatermarks.size(), i);

            while (iterator.hasNext()) {
                int[] dataUsersCombination = iterator.next();
                Arrays.sort(dataUsersCombination);
                List<Integer> dataUsers = new LinkedList<>();
                BigDecimal[] watermark = new BigDecimal[matchingFragment.getMeasurements().size()];
                Arrays.fill(watermark, BigDecimal.valueOf(0));

                for (int index : dataUsersCombination) {
```

```
                DataUserWatermark singleWatermark = singleWatermarks.get(index);
                dataUsers.add(singleWatermark.getDataUsers().get(0));

                for (int j = 0; j < watermark.length; j++) {
                    watermark[j] = watermark[j].add(singleWatermark.getWatermark()[j]);
                }
            }

            for (int j = 0; j < watermark.length; j++) {
                watermark[j] = watermark[j].divide(BigDecimal.valueOf(dataUsers.size()), 4,
                    RoundingMode.HALF_UP);
            }

            setOfWatermarks.add(new DataUserWatermark(dataUsers, watermark));
        }
    }
    return setOfWatermarks;
}

private static BigDecimal getWatermarkSimilarity(BigDecimal[] suspiciousWatermark, BigDecimal[]
        originalWatermark, UsabilityConstraint usabilityConstraint, HashMap<Integer, Integer>
        matchingMeasurements) {
    BigDecimal similarity = new BigDecimal("0.0");

    for (Entry<Integer, Integer> entry : matchingMeasurements.entrySet()) {
        BigDecimal suspiciousMark = suspiciousWatermark[entry.getKey()];
        BigDecimal originalMark = originalWatermark[entry.getValue()];

        BigDecimal distance = (suspiciousMark.subtract(originalMark)).abs();
        BigDecimal relativeDistance = distance.divide(
            usabilityConstraint.getMaximumError().multiply(BigDecimal.valueOf(2)), 4,
            RoundingMode.HALF_UP);

        BigDecimal markSimilarity = BigDecimal.valueOf(1).subtract(relativeDistance);
        if (markSimilarity.compareTo(BigDecimal.valueOf(0)) > 0
                && markSimilarity.compareTo(BigDecimal.valueOf(1)) < 1) {
            similarity = similarity.add(markSimilarity);
        }
    }

    similarity = similarity.divide(BigDecimal.valueOf(matchingMeasurements.size()), 4,
        RoundingMode.HALF_UP);
    return similarity;
}

private static List<DataUserWatermark> getDatasetLeakers(int datasetSize,
        List<List<DataUserWatermark>> setOfMatchingWatermarks) {
    List<DataUserWatermark> datasetLeakers = new LinkedList<>();

    for (int i = 0; i < setOfMatchingWatermarks.size(); i++) {
        List<DataUserWatermark> matchingWatermarks = setOfMatchingWatermarks.get(i);

        for (int j = 0; j < matchingWatermarks.size(); j++) {
            DataUserWatermark matchingWatermark = matchingWatermarks.get(j);

            if (datasetLeakers.contains(matchingWatermark)) {
                DataUserWatermark potentialLeaker =
                    datasetLeakers.get(datasetLeakers.indexOf(matchingWatermark));
                potentialLeaker
                    .setProbability(potentialLeaker.getProbability().add(matchingWatermark.getProbability()));
            } else {
                datasetLeakers.add(matchingWatermark);
            }

        }
    }

    for (int i = 0; i < datasetLeakers.size(); i++) {
        DataUserWatermark dataLeaker = datasetLeakers.get(i);
        dataLeaker.setProbability(
                dataLeaker.getProbability().divide(BigDecimal.valueOf(datasetSize), 4, RoundingMode.HALF_UP));
        datasetLeakers.get(i).setProbability(dataLeaker.getProbability());
    }
    return datasetLeakers;
}
}
```

watermarking/src/main/java/simulators/DataDetectiveSimulator.java

```java
package simulators;

import java.math.BigDecimal;

import services.DetectionService;
import utilities.LogService;
import utilities.TimeService;

public class DataDetectiveSimulator {

   public static void detectLeakage(String datasetName, Double fragmentSimilarityThreshold,
         Double watermarkSimilarityThreshold, int numberOfColluders) {
      LogService.log(LogService.SIMULATOR_LEVEL, "DataDetectiveSimulator",
            "detectLeakage(datasetName=" + datasetName + ", fragmentSimilarityThreshold="
            + fragmentSimilarityThreshold + ", watermarkSimilarityThreshold=" + watermarkSimilarityThreshold
            + ")");
      TimeService timeService = new TimeService();
      String reportName = datasetName.substring(0, datasetName.indexOf(".json")) + "_report.txt";
      DetectionService.detectLeakage(datasetName, reportName, BigDecimal.valueOf(fragmentSimilarityThreshold),
            BigDecimal.valueOf(watermarkSimilarityThreshold), numberOfColluders);
      timeService.stop();
      LogService.log(LogService.SIMULATOR_LEVEL, "DataDetectiveSimulator", "detectLeakage",
         timeService.getTime());
   }
}
```

watermarking/src/main/java/simulators/DataUserSimulator.java

```java
package simulators;

import java.math.BigDecimal;
import java.math.RoundingMode;
import java.time.LocalDate;
import java.util.LinkedList;
import java.util.List;
import java.util.Random;

import entities.Fragment;
import entities.Measurement;
import services.DataService;
import utilities.FileService;
import utilities.FragmentationService;
import utilities.LogService;
import utilities.TimeService;

public class DataUserSimulator {

   public static void requestDataset(int dataUserId, String deviceId, String from, String to) {
      LogService.log(LogService.SIMULATOR_LEVEL, "DataUserSimulator", "requestDataset(dataUserId=" +
         dataUserId + ", deviceId=" + deviceId + ", from=" + from + ", to=" + to + ")");
      TimeService timeService = new TimeService();
      String datasetName = "requestedDataset_by" + dataUserId + "_" + deviceId + "_" + from.toString() + "_"
            + to.toString() + ".json";
      DataService.requestDataset(datasetName, dataUserId, deviceId, "cbg", "mmol/L", LocalDate.parse(from),
            LocalDate.parse(to));
      timeService.stop();
      LogService.log(LogService.SIMULATOR_LEVEL, "DataUserSimulator", "requestDataset",
         timeService.getTime());
   }

   public static void requestDataset(int dataUserId, int numberOfDevices, String from, String to) {
      LogService.log(LogService.SIMULATOR_LEVEL, "DataUserSimulator", "requestDataset(dataUserId=" +
         dataUserId + ", noOfDevices=" + numberOfDevices + ", from=" + from + ", to=" + to + ")");
      TimeService timeService = new TimeService();
      String datasetName = "requestedDataset_by" + dataUserId + "_" + numberOfDevices + "_" + from.toString()
         + "_" + to.toString() + ".json";
      DataService.requestDataset(datasetName, dataUserId, numberOfDevices, "cbg", "mmol/L",
         LocalDate.parse(from), LocalDate.parse(to));
      timeService.stop();
      LogService.log(LogService.SIMULATOR_LEVEL, "DataUserSimulator", "requestDataset",
         timeService.getTime());
   }

   public static void attackDatasetByDeletion(String datasetName, int n) {
      List<Fragment> dataset = FragmentationService.getFragments(datasetName);
      List<Fragment> newDataset = new LinkedList<>();

      for (int i = 0; i < dataset.size(); i++) {
         Fragment fragment = dataset.get(i);
         Fragment newFragment = new Fragment(fragment.getDeviceId(), fragment.getType(), fragment.getUnit(),
               fragment.getDate());

         for (int j = 0; j < fragment.getMeasurements().size(); j++) {
            if (j % n != 0) {
               newFragment.getMeasurements().add(fragment.getMeasurements().get(j));
            }
         }

         newDataset.add(newFragment);
      }

      String attackedDatasetName = datasetName.substring(0, datasetName.indexOf(".json")) + "_deletion_" + n
            + ".json";
      FileService.writeDataset(attackedDatasetName, newDataset);
   }

   public static void attackDatasetByRounding(String datasetName, int decimalDigit) {
      List<Fragment> dataset = FragmentationService.getFragments(datasetName);

      for (int i = 0; i < dataset.size(); i++) {
         Fragment fragment = dataset.get(i);

         for (int j = 0; j < fragment.getMeasurements().size(); j++) {
            BigDecimal roundedValue = fragment.getMeasurements().get(j).getValue().setScale(decimalDigit,
                  RoundingMode.HALF_UP);
            fragment.getMeasurements().get(j).setValue(roundedValue);
         }

         dataset.set(i, fragment);
      }

      String attackedDatasetName = datasetName.substring(0, datasetName.indexOf(".json")) + "_rounding_"
            + decimalDigit + ".json";
      FileService.writeDataset(attackedDatasetName, dataset);
  }

  public static void attackDatasetbyRandom(String datasetName, Double maxError, Long seed) {
      Random random = new Random(seed);
      List<Fragment> dataset = FragmentationService.getFragments(datasetName);

      for (int i = 0; i < dataset.size(); i++) {
         Fragment fragment = dataset.get(i);

         for (int j = 0; j < fragment.getMeasurements().size(); j++) {
            BigDecimal randomValue = BigDecimal.valueOf(random.nextDouble() * maxError);
            BigDecimal newValue = fragment.getMeasurements().get(j).getValue();

            if (random.nextBoolean() == true) {
                  newValue = newValue.add(randomValue);
            } else {
                  newValue = newValue.subtract(randomValue);
            }

            fragment.getMeasurements().get(j).setValue(newValue);
         }

         dataset.set(i, fragment);
      }

      String attackedDatasetName = datasetName.substring(0, datasetName.indexOf(".json")) + "_random_"
            + maxError.toString().replace(".", "-") + ".json";
      FileService.writeDataset(attackedDatasetName, dataset);
  }

  public static void attackDatasetbySubset(String datasetName, int startIndex, int endIndex) {
      List<Fragment> dataset = FragmentationService.getFragments(datasetName);
      List<Fragment> newDataset = new LinkedList<>();

      for (int i = 0; i < dataset.size(); i++) {
         Fragment fragment = dataset.get(i);
         Fragment newFragment = new Fragment(fragment.getDeviceId(), fragment.getType(), fragment.getUnit(),
                  fragment.getDate());

         for (int j = 0; j < fragment.getMeasurements().size(); j++) {
            if (j >= startIndex && j <= endIndex) {
                  newFragment.getMeasurements().add(fragment.getMeasurements().get(j));
            }
         }

         newDataset.add(newFragment);
      }

      String attackedDatasetName = datasetName.substring(0, datasetName.indexOf(".json")) + "_subset_" +
         startIndex + "-" + endIndex + ".json";
      FileService.writeDataset(attackedDatasetName, newDataset);
  }

  public static void attackDatasetByCollusion(List<String> datasetNames) {
      List<List<Fragment>> datasets = new LinkedList<>();
      String attackedDatasetName = "colludedDataset_by";

      for (String datasetName : datasetNames) {
         datasets.add(FragmentationService.getFragments(datasetName));
         attackedDatasetName = attackedDatasetName + "-" + datasetName.split("_")[1].substring(2);
      }

      List<Fragment> dataset = new LinkedList<>();
      for (int i = 0; i < datasets.get(0).size(); i++) {
         Fragment newFragment = new Fragment(datasets.get(0).get(i).getDeviceId(),
            datasets.get(0).get(i).getType(), datasets.get(0).get(i).getUnit(),
            datasets.get(0).get(i).getDate());
         for (int j = 0; j < datasets.get(0).get(i).getMeasurements().size(); j++) {
            BigDecimal newValue = new BigDecimal("0.0");

            for (List<Fragment> colludingDataset : datasets) {
               newValue = newValue.add(colludingDataset.get(i).getMeasurements().get(j).getValue());
            }

            newValue = newValue.divide(BigDecimal.valueOf(datasets.size()), 15, RoundingMode.HALF_UP);
            Measurement newMeasurement = new Measurement(newFragment.getDeviceId(), newFragment.getType(),
                  newFragment.getUnit(), datasets.get(0).get(i).getMeasurements().get(j).getTime(),
                  newValue);
            newFragment.getMeasurements().add(newMeasurement);
         }

         dataset.add(newFragment);
      }

      attackedDatasetName = attackedDatasetName + ".json";
      FileService.writeDataset(attackedDatasetName, dataset);
   }
}
```

watermarking/src/main/java/simulators/PatientSimulator.java

```java
package simulators;

import java.math.BigDecimal;
import java.time.LocalDate;
import java.time.LocalDateTime;
import java.util.LinkedList;
import java.util.List;
import java.util.Random;

import entities.Fragment;
import entities.Measurement;
import services.ContainerService;
import utilities.FileService;
import utilities.LogService;
import utilities.TimeService;

public class PatientSimulator {

   public static void storeDataset(Boolean randomSecretKey, String datasetName) {
      LogService.log(LogService.SIMULATOR_LEVEL, "PatientSimulator", "storeDataset(datasetName=" + datasetName
+ ")");
      TimeService timeService = new TimeService();
      ContainerService.storeDataset(randomSecretKey, datasetName);
      timeService.stop();
      LogService.log(LogService.SIMULATOR_LEVEL, "PatientSimulator", "storeDataset", timeService.getTime());
   }

   public static void generateDataset(String deviceId, String from, String to, Long seed) {
      List<Fragment> dataset = new LinkedList<>();
      Random random = new Random(seed);

      int minute = 0;
      int second = 0;
      BigDecimal value = BigDecimal.valueOf(random.nextDouble() * 22.5);

      int sequenceLength = 0;
      char sequenceSign = '+';
      if (random.nextBoolean() == false) {
         sequenceSign = '-';
      }

      LocalDate startDate = LocalDate.parse(from);
      LocalDate endDate = LocalDate.parse(to);

      while (!startDate.isAfter(endDate)) {
         Fragment fragment = new Fragment();
         fragment.setMeasurements(new LinkedList<>());

         int hour = 0;
         while (hour < 24) {
            LocalDateTime time = LocalDateTime.of(startDate.getYear(), startDate.getMonth(),
                  startDate.getDayOfMonth(), hour, minute, second);

            if (sequenceLength == 0) {
               if (random.nextBoolean() == true) {
                  sequenceLength = random.nextInt(20);
               } else {
                  sequenceLength = random.nextInt(40);
               }
               if (sequenceSign == '+') {
                  sequenceSign = '-';
               } else {
                  sequenceSign = '+';
               }
            } else {
               sequenceLength = sequenceLength - 1;
            }

            BigDecimal valueChange = BigDecimal.valueOf(random.nextDouble());
            if (sequenceSign == '+') {
               value = value.add(valueChange);
               if (value.compareTo(BigDecimal.valueOf(55.0)) >= 0) {
                  value = value.subtract(valueChange).subtract(valueChange);
                  sequenceLength = 0;
               }
            } else {
                value = value.subtract(valueChange);
                if (value.compareTo(BigDecimal.valueOf(0.0)) <= 0) {
                   value = value.add(valueChange).add(valueChange);
                   sequenceLength = 0;
                }
             }

             Measurement measurement = new Measurement(deviceId, "cbg", "mmol/L", time, value);
             fragment.getMeasurements().add(measurement);

             minute = minute + 5;
             if (minute >= 60) {
                hour = hour + 1;
                minute = minute - 60;
             }
          }

          dataset.add(fragment);
          startDate = startDate.plusDays(1);
       }

       FileService.writeDataset("generatedDataset_" + deviceId + "_" + from.toString() + "_" + to.toString() +
".json",
             dataset);
    }

}
```

watermarking/src/main/java/ui/ConsoleUI.java

```java
package ui;

import java.util.LinkedList;
import java.util.List;

import entities.UsabilityConstraint;
import simulators.DataDetectiveSimulator;
import simulators.DataUserSimulator;
import simulators.PatientSimulator;
import utilities.DatabaseService;
import utilities.FileService;
import utilities.LogService;

public class ConsoleUI {
   public static void main(String[] args) {

      FileService.FOLDER = "C:/temp/files/";

      if (args.length == 0) {
         System.out.println("Commands");
         System.out.println("-reset -table -fragment");
         System.out.println("-reset -table -request");
         System.out.println("-reset -table -usability_constraint");
         System.out.println("-reset -files");
         System.out.println("-reset -log");
         System.out.println("");
         System.out.println("-set -usability_constraint [maximumError] [numberOfRanges]");
         System.out.println("");
         System.out.println("-generate [deviceId] [from] [to] [seed]");
         System.out.println("");
         System.out.println("-store -random [datasetName]");
         System.out.println("-store -one [datasetName]");
         System.out.println("");
         System.out.println("-request -patient [dataUserId] [deviceId] [from] [to]");
         System.out.println("-request -patients [dataUserId] [numberOfDevices] [from] [to]");
         System.out.println("");
         System.out.println("-attack -deletion [datasetName] [frequency]");
         System.out.println("-attack -random [datasetName] [maximumError] [seed]");
         System.out.println("-attack -rounding [datasetName] [decimalDigit]");
         System.out.println("-attack -subset [datasetName] [startIndex] [endIndex]");
         System.out.println("-attack -collusion [datasetName1] ... [datasetNameN]");
         System.out.println("");
         System.out.println(
               "-detect [datasetName] [fragmentSimilarityThreshold] [watermarkSimilarityThreshold]
               [numberOfColluders]");
         System.out.println("");
         System.out.println("Configurable Parameters");
         System.out.println("[dataUserId]:\t\t\tdata user identifier as integer");
         System.out.println("[datasetName]:\t\t\tdataset name as string");
         System.out.println("[decimalDigit]:\t\t\tdecimal digit as integer");
         System.out.println("[deviceId]:\t\t\tdevice identifier as string");
         System.out.println("[endIndex]:\t\t\tend index as integer");
         System.out.println("[fragmentSimilarityThreshold]:\tfragment similarity threshold as double");
         System.out.println("[frequency]:\t\t\tfrequency as integer");
         System.out.println("[from]:\t\t\t\tstart date as string with the format [yyyy-MM-dd]");
         System.out.println("[maximumError]:\t\t\tmaximum error as double");
         System.out.println("[numberOfColluders]:\t\tnumber of colluders as integer");
         System.out.println("[numberOfDevices]:\t\tnumber of devices as integer");
         System.out.println("[numberOfRanges]:\t\tnumber of ranges as integer");
         System.out.println("[seed]:\t\t\t\tseed as integer");
         System.out.println("[startIndex]:\t\t\tstart index as integer");
         System.out.println("[to]:\t\t\t\tend date as string with the format [yyyy-MM-dd]");
         System.out.println("[watermarkSimilarityThreshold]:\twatermark similarity threshold as double");
      }

      else {
         if (args[0].contentEquals("-reset")) {
            if (args[1].contentEquals("-table")) {
               if (args[2].contentEquals("-fragment")) {
                  DatabaseService.deleteTable("fragment");
               }
               if (args[2].contentEquals("-usability_constraint")) {
                  DatabaseService.deleteTable("usability_constraint");
               }
               if (args[2].contentEquals("-request")) {
                  DatabaseService.deleteTable("request");
```

```
                }
            }
            if (args[1].contentEquals("-files")) {
                FileService.deleteFiles();
            }
            if (args[1].contentEquals("-log")) {
                LogService.delete();
            }
        }
        if (args[0].contentEquals("-set")) {
            if (args[1].contentEquals("-usability_constraint")) {
                DatabaseService.insertUsabilityConstraint(
                        new UsabilityConstraint(Double.parseDouble(args[2]), Integer.parseInt(args[3])));
            }
        }
        if (args[0].contentEquals("-generate")) {
            PatientSimulator.generateDataset(args[1], args[2], args[3], Long.parseLong(args[4]));
        }
        if (args[0].contentEquals("-store")) {
            if (args[1].contentEquals("-one")) {
                PatientSimulator.storeDataset(false, args[2]);
            }
            if (args[1].contentEquals("-random")) {
                PatientSimulator.storeDataset(true, args[2]);
            }
        }
        if (args[0].contentEquals("-request")) {
            if (args[1].contentEquals("-patient")) {
                DataUserSimulator.requestDataset(Integer.parseInt(args[2]), args[3], args[4], args[5]);
            }
            if (args[1].contentEquals("-patients")) {
                DataUserSimulator.requestDataset(Integer.parseInt(args[2]), Integer.parseInt(args[3]),
                    args[4], args[5]);
            }
        }
        if (args[0].contentEquals("-attack")) {
            if (args[1].contentEquals("-deletion")) {
                DataUserSimulator.attackDatasetByDeletion(args[2], Integer.parseInt(args[3]));
            }
            if (args[1].contentEquals("-random")) {
                DataUserSimulator.attackDatasetbyRandom(args[2], Double.parseDouble(args[3]),
                        Long.parseLong(args[4]));
            }
            if (args[1].contentEquals("-rounding")) {
                DataUserSimulator.attackDatasetByRounding(args[2], Integer.parseInt(args[3]));
            }
            if (args[1].contentEquals("-subset")) {
                DataUserSimulator.attackDatasetbySubset(args[2], Integer.parseInt(args[3]),
                        Integer.parseInt(args[4]));
            }
            if (args[1].contentEquals("-collusion")) {
                List<String> datasetNames = new LinkedList<>();
                for (int i = 2; i < args.length; i++) {
                    datasetNames.add(args[i]);
                }
                DataUserSimulator.attackDatasetByCollusion(datasetNames);
            }
        }
        if (args[0].contentEquals("-detect")) {
            DataDetectiveSimulator.detectLeakage(args[1], Double.parseDouble(args[2]),
                Double.parseDouble(args[3]), Integer.parseInt(args[4]));
        }
    }
  }
}
```

watermarking/src/main/java/ui/ProgramUI.java

```
package ui;

import entities.UsabilityConstraint;
import simulators.DataDetectiveSimulator;
import simulators.DataUserSimulator;
import simulators.PatientSimulator;
import utilities.DatabaseService;
import utilities.FileService;
import utilities.LogService;

public class ProgramUI {

   public static void main(String[] args) {

      DatabaseService.deleteTable("fragment");
      DatabaseService.deleteTable("request");
      DatabaseService.deleteTable("usability_constraint");
      FileService.deleteFiles("Dataset");
      LogService.delete();

      DatabaseService.insertUsabilityConstraint(new UsabilityConstraint(0.5, 10));

      PatientSimulator.storeDataset(false, "testdata.json");

      DataUserSimulator.requestDataset(1, "DexG5MobRec_SM64305440", "2017-02-04", "2017-02-05");
      DataUserSimulator.requestDataset(2, "DexG5MobRec_SM64305440", "2017-02-04", "2017-02-04");
      DataDetectiveSimulator.detectLeakage("requestedDataset_by1_DexG5MobRec_SM64305440_2017-02-04_2017-02-
         05.json", 0.01, 0.01, 2);
   }
}
```

watermarking/src/main/java/utilities/DatabaseService.java

```java
package utilities;

import java.sql.Connection;
import java.sql.Date;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.sql.Timestamp;
import java.time.LocalDate;
import java.time.LocalDateTime;
import java.util.ArrayList;
import java.util.Collections;
import java.util.LinkedList;
import java.util.List;

import entities.Fragment;
import entities.Request;
import entities.UsabilityConstraint;

public class DatabaseService {

   public static void deleteTable(String table) {
      try (Connection connection =
         DriverManager.getConnection("jdbc:postgresql://localhost:5432/watermarking", "postgres", "admin")) {
         String sql = "DELETE FROM " + table;

         PreparedStatement preparedStatement = connection.prepareStatement(sql);
         preparedStatement.executeUpdate();
         preparedStatement.close();
         connection.close();
      } catch (SQLException ex) {
         ex.printStackTrace();
      }
   }

   public static void insertUsabilityConstraint(UsabilityConstraint usabilityConstraint) {
      try (Connection connection =
         DriverManager.getConnection("jdbc:postgresql://localhost:5432/watermarking", "postgres", "admin")) {
         String sql = "INSERT INTO usability_constraint (type, unit, minimum_value, maximum_value,
            maximum_error, number_of_ranges) VALUES (?, ?, ?, ?, ?, ?)";

         PreparedStatement preparedStatement = connection.prepareStatement(sql);
         preparedStatement.setString(1, usabilityConstraint.getType());
         preparedStatement.setString(2, usabilityConstraint.getUnit());
         preparedStatement.setBigDecimal(3, usabilityConstraint.getMinimumValue());
         preparedStatement.setBigDecimal(4, usabilityConstraint.getMaximumValue());
         preparedStatement.setBigDecimal(5, usabilityConstraint.getMaximumError());
         preparedStatement.setInt(6, usabilityConstraint.getNumberOfRanges());
         preparedStatement.executeUpdate();
         preparedStatement.close();
         connection.close();
      } catch (SQLException ex) {
         ex.printStackTrace();
      }
   }

   public static UsabilityConstraint getUsabilityConstraint(String type, String unit) {
      try (Connection connection =
         DriverManager.getConnection("jdbc:postgresql://localhost:5432/watermarking", "postgres", "admin")) {
         String sql = "SELECT * FROM usability_constraint WHERE type = ? AND unit = ?";

         PreparedStatement preparedStatement = connection.prepareStatement(sql);
         preparedStatement.setString(1, type);
         preparedStatement.setString(2, unit);

         ResultSet resultSet = preparedStatement.executeQuery();
         if (resultSet.next()) {
            return new UsabilityConstraint(type, unit, resultSet.getBigDecimal("minimum_value"),
                  resultSet.getBigDecimal("maximum_value"), resultSet.getBigDecimal("maximum_error"),
                  resultSet.getInt("number_of_ranges"));
         }

         resultSet.close();
         preparedStatement.close();
         connection.close();
```

```
        } catch (Exception ex) {
            ex.printStackTrace();
        }
        return null;
    }

    public static void insertFragments(List<Fragment> fragments) {
        try (Connection connection =
            DriverManager.getConnection("jdbc:postgresql://localhost:5432/watermarking", "postgres", "admin")) {
            String sql = "INSERT INTO fragment (device_id, type, unit, date, measurements, secret_key) "
                    + "VALUES (?, ?, ?, ?, ?::JSON, ?)";

            PreparedStatement preparedStatement = connection.prepareStatement(sql);
            for (Fragment fragment : fragments) {
                preparedStatement.setString(1, fragment.getDeviceId());
                preparedStatement.setString(2, fragment.getType());
                preparedStatement.setString(3, fragment.getUnit());
                preparedStatement.setDate(4, Date.valueOf(fragment.getDate()));
                preparedStatement.setObject(5, fragment.getMeasurementsAsJsonArrayString());
                preparedStatement.setLong(6, fragment.getSecretKey());
                preparedStatement.executeUpdate();
            }

            preparedStatement.close();
            connection.close();
        } catch (SQLException ex) {
            ex.printStackTrace();
        }
    }

    public static Fragment getFragment(String deviceId, String type, String unit, LocalDate date) {
        try (Connection connection =
            DriverManager.getConnection("jdbc:postgresql://localhost:5432/watermarking", "postgres", "admin")) {
            String sql = "SELECT date, measurements, secret_key FROM fragment "
                    + "WHERE device_id = ? AND type = ? AND unit = ? AND date = ?";

            PreparedStatement preparedStatement = connection.prepareStatement(sql);
            preparedStatement.setString(1, deviceId);
            preparedStatement.setString(2, type);
            preparedStatement.setString(3, unit);
            preparedStatement.setDate(4, Date.valueOf(date));

            ResultSet resultSet = preparedStatement.executeQuery();
            if (resultSet.next()) {
                Fragment fragment = new Fragment(deviceId, type, unit,
                    LocalDate.parse(resultSet.getString("date")), resultSet.getLong("secret_key"));
                fragment.setMeasurementsFromJsonArrayString(resultSet.getString("measurements"));
                Collections.sort(fragment.getMeasurements());
                return fragment;
            }

            resultSet.close();
            preparedStatement.close();
            connection.close();
        } catch (Exception ex) {
            ex.printStackTrace();
        }
        return null;
    }

    public static List<Fragment> getFragments(String deviceId, String type, String unit, LocalDate from,
            LocalDate to) {
        List<Fragment> fragments = new LinkedList<>();
        try (Connection connection =
            DriverManager.getConnection("jdbc:postgresql://localhost:5432/watermarking", "postgres", "admin")) {
            String sql = "SELECT date, measurements, secret_key FROM fragment "
                    + "WHERE device_id = ? AND type = ? AND unit = ? AND date BETWEEN ? AND ?";

            PreparedStatement preparedStatement = connection.prepareStatement(sql);
            preparedStatement.setString(1, deviceId);
            preparedStatement.setString(2, type);
            preparedStatement.setString(3, unit);
            preparedStatement.setDate(4, Date.valueOf(from));
            preparedStatement.setDate(5, Date.valueOf(to));

            ResultSet resultSet = preparedStatement.executeQuery();
            while (resultSet.next()) {
```

```
            Fragment fragment = new Fragment(deviceId, type, unit,
               LocalDate.parse(resultSet.getString("date")), resultSet.getLong("secret_key"));
            fragment.setMeasurementsFromJsonArrayString(resultSet.getString("measurements"));
            Collections.sort(fragment.getMeasurements());
            fragments.add(fragment);
         }

         resultSet.close();
         preparedStatement.close();
         connection.close();
      } catch (Exception ex) {
         ex.printStackTrace();
      }

      Collections.sort(fragments);
      return fragments;
   }

   public static List<Fragment> getFragments(String type, String unit) {
      List<Fragment> fragments = new LinkedList<>();
      try (Connection connection =
         DriverManager.getConnection("jdbc:postgresql://localhost:5432/watermarking", postgres", "admin")) {
         String sql = "SELECT * FROM fragment WHERE type = ? AND unit = ?";

         PreparedStatement preparedStatement = connection.prepareStatement(sql);
         preparedStatement.setString(1, type);
         preparedStatement.setString(2, unit);

         ResultSet resultSet = preparedStatement.executeQuery();
         while (resultSet.next()) {
            Fragment fragment = new Fragment(resultSet.getString("device_id"), type, unit,
                  LocalDate.parse(resultSet.getString("date")), resultSet.getLong("secret_key"));
            fragment.setMeasurementsFromJsonArrayString(resultSet.getString("measurements"));
            Collections.sort(fragment.getMeasurements());
            fragments.add(fragment);
         }

         resultSet.close();
         preparedStatement.close();
         connection.close();
      } catch (Exception ex) {
         ex.printStackTrace();
      }

      Collections.sort(fragments);
      return fragments;
   }

   public static List<Fragment> getFragments(int noOfDevices, String type, String unit, LocalDate from,
         LocalDate to) {
      List<Fragment> fragments = new LinkedList<>();
      try (Connection connection =
         DriverManager.getConnection("jdbc:postgresql://localhost:5432/watermarking", "postgres", "admin")) {
         String sql = "SELECT distinct device_id FROM fragment "
               + "WHERE type = ? AND unit = ? AND date BETWEEN ? AND ?";

         PreparedStatement preparedStatement = connection.prepareStatement(sql);
         preparedStatement.setString(1, type);
         preparedStatement.setString(2, unit);
         preparedStatement.setDate(3, Date.valueOf(from));
         preparedStatement.setDate(4, Date.valueOf(to));

         ResultSet resultSet = preparedStatement.executeQuery();
         int counter = 0;
         while (resultSet.next() && counter < noOfDevices) {
            fragments.addAll(getFragments(resultSet.getString("device_id"), type, unit, from, to));
            counter = counter + 1;
         }

         resultSet.close();
         preparedStatement.close();
         connection.close();
      } catch (Exception ex) {
         ex.printStackTrace();
      }

      return fragments;
```

```
    }

    public static void insertRequest(Request request) {
        try (Connection connection =
            DriverManager.getConnection("jdbc:postgresql://localhost:5432/watermarking", "postgres", "admin")) {
            String sql = "INSERT INTO request (device_id, data_user, type, unit, date, number_of_watermark,
                timestamps) " + "VALUES (?, ?, ?, ?, ?, ?, ?)";

            PreparedStatement preparedStatement = connection.prepareStatement(sql);
            preparedStatement.setString(1, request.getDeviceId());
            preparedStatement.setInt(2, request.getDataUser());
            preparedStatement.setString(3, request.getType());
            preparedStatement.setString(4, request.getUnit());
            preparedStatement.setDate(5, Date.valueOf(request.getDate()));
            preparedStatement.setInt(6, request.getNumberOfWatermark());
            preparedStatement.setArray(7, connection.createArrayOf("timestamp",
                request.getTimestamps().toArray()));
            preparedStatement.executeUpdate();

            preparedStatement.close();
            connection.close();
        } catch (SQLException ex) {
            ex.printStackTrace();
        }
    }

    public static Request getRequest(int dataUser, String deviceId, String type, String unit, LocalDate date) {
        try (Connection connection =
            DriverManager.getConnection("jdbc:postgresql://localhost:5432/watermarking", "postgres", "admin")) {
            String sql = "SELECT number_of_watermark, timestamps FROM request WHERE device_id = ? AND data_user =
                ? AND type = ? AND unit = ? AND date = ?";

            PreparedStatement preparedStatement = connection.prepareStatement(sql);
            preparedStatement.setString(1, deviceId);
            preparedStatement.setInt(2, dataUser);
            preparedStatement.setString(3, type);
            preparedStatement.setString(4, unit);
            preparedStatement.setDate(5, Date.valueOf(date));

            ResultSet resultSet = preparedStatement.executeQuery();
            if (resultSet.next()) {
                Timestamp[] timestampArray = (Timestamp[]) resultSet.getArray("timestamps").getArray();
                ArrayList<LocalDateTime> timestamps = new ArrayList<LocalDateTime>();
                for (int i = 0; i < timestampArray.length; i++) {
                    timestamps.add(timestampArray[i].toLocalDateTime());
                }
                return new Request(deviceId, dataUser, type, unit, date, resultSet.getInt("number_of_watermark"),
                        timestamps);
            }

            resultSet.close();
            preparedStatement.close();
            connection.close();
        } catch (Exception ex) {
            ex.printStackTrace();
        }
        return null;
    }

    public static List<Request> getRequests(String deviceId, String type, String unit, LocalDate date) {
        List<Request> requests = new LinkedList<>();
        try (Connection connection =
            DriverManager.getConnection("jdbc:postgresql://localhost:5432/watermarking", "postgres", "admin")) {
            String sql = "SELECT data_user, number_of_watermark, timestamps FROM request "
                    + "WHERE device_id = ? AND type = ? AND unit = ? AND date = ?";

            PreparedStatement preparedStatement = connection.prepareStatement(sql);
            preparedStatement.setString(1, deviceId);
            preparedStatement.setString(2, type);
            preparedStatement.setString(3, unit);
            preparedStatement.setDate(4, Date.valueOf(date));

            ResultSet resultSet = preparedStatement.executeQuery();
            while (resultSet.next()) {
                Timestamp[] timestampArray = (Timestamp[]) resultSet.getArray("timestamps").getArray();
                ArrayList<LocalDateTime> timestamps = new ArrayList<LocalDateTime>();
                for (int i = 0; i < timestampArray.length; i++) {
```

```
                timestamps.add(timestampArray[i].toLocalDateTime());
            }
            requests.add(new Request(deviceId, resultSet.getInt("data_user"), type, unit, date,
                    resultSet.getInt("number_of_watermark"), timestamps));
        }

        resultSet.close();
        preparedStatement.close();
        connection.close();
    } catch (Exception ex) {
        ex.printStackTrace();
    }

    return requests;
}

public static void updateRequest(Request request) {
    try (Connection connection =
        DriverManager.getConnection("jdbc:postgresql://localhost:5432/watermarking", "postgres", "admin")) {
        String sql = "UPDATE request SET timestamps = ? WHERE device_id = ? AND data_user = ? AND type = ? "
                + "AND unit = ? AND date = ?";

        PreparedStatement preparedStatement = connection.prepareStatement(sql);
        preparedStatement.setArray(1, connection.createArrayOf("timestamp",
            request.getTimestamps().toArray()));
        preparedStatement.setString(2, request.getDeviceId());
        preparedStatement.setInt(3, request.getDataUser());
        preparedStatement.setString(4, request.getType());
        preparedStatement.setString(5, request.getUnit());
        preparedStatement.setDate(6, Date.valueOf(request.getDate()));
        preparedStatement.executeUpdate();

        preparedStatement.close();
        connection.close();
    } catch (SQLException ex) {
        ex.printStackTrace();
    }
}

public static int getNumberOfWatermark(String deviceId, String type, String unit, LocalDate date) {
    try (Connection connection =
        DriverManager.getConnection("jdbc:postgresql://localhost:5432/watermarking", "postgres", "admin")) {
        String sql = "SELECT max(number_of_watermark) FROM request WHERE device_id = ? AND type = ? "
                + "AND unit = ? AND date = ?";

        PreparedStatement preparedStatement = connection.prepareStatement(sql);
        preparedStatement.setString(1, deviceId);
        preparedStatement.setString(2, type);
        preparedStatement.setString(3, unit);
        preparedStatement.setDate(4, Date.valueOf(date));

        ResultSet resultSet = preparedStatement.executeQuery();
        if (resultSet.next()) {
            return resultSet.getInt("max");
        }

        resultSet.close();
        preparedStatement.close();
        connection.close();
    } catch (Exception ex) {
        ex.printStackTrace();
    }
    return 0;
}
}
```

watermarking/src/main/java/utilities/FileService.java

```java
package utilities;

import java.io.BufferedWriter;
import java.io.File;
import java.io.FileWriter;
import java.io.IOException;
import java.util.List;

import entities.Fragment;

public class FileService {

   public static String FOLDER = "files/";

   public static void writeDataset(String datasetName, List<Fragment> dataset) {
      String jsonString = "[";
      for (int i = 0; i < dataset.size(); i++) {
         jsonString = jsonString + dataset.get(i).getMeasurementsAsJsonString();
         if (i + 1 < dataset.size())
            jsonString = jsonString + ",";
      }
      jsonString = jsonString + "\n]";
      writeFile(datasetName, jsonString);
   }

   public static void writeLine(String fileName, String line) {
      try (BufferedWriter out = new BufferedWriter(new FileWriter(FileService.FOLDER + fileName, true))) {
         out.write(line + "\n");
      } catch (IOException ex) {
         ex.printStackTrace();
      }
   }

   public static void writeFile(String fileName, String message) {
      try (FileWriter file = new FileWriter(FOLDER + fileName)) {
         file.write(message);
      } catch (IOException ex) {
         ex.printStackTrace();
      }
   }

   public static void deleteFiles() {
      for (File file : new File(FOLDER).listFiles()) {
         if (file.getName().contains("Dataset") || file.getName().contains("report")) {
            file.delete();
         }
      }
   }

   public static void deleteFiles(String name) {
      for (File file : new File(FOLDER).listFiles()) {
         if (file.getName().contains(name)) {
            file.delete();
         }
      }
   }
}
```

watermarking/src/main/java/utilities/FragmentationService.java

```java
package utilities;

import java.io.FileReader;
import java.io.IOException;
import java.math.BigDecimal;
import java.time.LocalDateTime;
import java.time.format.DateTimeFormatter;
import java.util.Collections;
import java.util.LinkedList;
import java.util.List;

import org.json.simple.JSONArray;
import org.json.simple.JSONObject;
import org.json.simple.parser.JSONParser;
import org.json.simple.parser.ParseException;

import entities.Fragment;
import entities.Measurement;

public class FragmentationService {

   public static List<Fragment> getFragments(String fileName) {
      List<Fragment> fragments = new LinkedList<>();

      try {
         DateTimeFormatter formatter = DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm:ss");
         JSONParser parser = new JSONParser();

         JSONArray array = (JSONArray) parser.parse(new FileReader(FileService.FOLDER + fileName));
         for (Object measurementObject : array) {
            JSONObject jsonMeasurement = (JSONObject) measurementObject;

            String deviceId = (String) jsonMeasurement.get("deviceId");
            String type = (String) jsonMeasurement.get("type");
            String unit = (String) jsonMeasurement.get("unit");
            String timeString = (String) jsonMeasurement.get("time");

            if (timeString.contains("T")) {
               timeString = timeString.replace("T", " ");
            }
            if (timeString.contains(".")) {
               timeString = timeString.split("\\.")[0];

            }
            LocalDateTime time = LocalDateTime.parse(timeString, formatter);
            BigDecimal value = new BigDecimal(jsonMeasurement.get("value").toString());
            Measurement measurement = new Measurement(deviceId, type, unit, time, value);

            Fragment fragment = new Fragment(measurement.getDeviceId(), measurement.getType(),
                  measurement.getUnit(), measurement.getTime().toLocalDate());
            if (fragments.contains(fragment)) {
               fragments.get(fragments.indexOf(fragment)).getMeasurements().add(measurement);
            } else {
               fragment.getMeasurements().add(measurement);
               fragments.add(fragment);
            }
         }

         Collections.sort(fragments);
         for (Fragment fragment : fragments) {
            Collections.sort(fragment.getMeasurements());
         }
      } catch (IOException ex) {
         ex.printStackTrace();
      } catch (ParseException ex) {
         ex.printStackTrace();
      }
      return fragments;
   }
}
```

watermarking/src/main/java/utilities/LogService.java

```java
package utilities;

import java.time.LocalDateTime;
import java.time.format.DateTimeFormatter;

public class LogService {

   public static String SIMULATOR_LEVEL = "\t";
   public static String SERVICE_LEVEL = "\t\t";
   private static DateTimeFormatter formatter = DateTimeFormatter.ofPattern("yyyy-MM-dd HH:mm:ss");
   private static final String logName = "log.txt";

   public static void log(String level, String className, String operation) {
      FileService.writeLine(logName,
            LocalDateTime.now().format(formatter) + level + "[" + className + "]\t" + operation + ": ...");
   }

   public static void log(String level, String className, String operation, Long seconds) {
      FileService.writeLine(logName, LocalDateTime.now().format(formatter) + level + "[" + className + "]\t"
            + operation + ": " + seconds + "ms.");
      if (level.contentEquals(SIMULATOR_LEVEL)) {
         FileService.writeLine(logName, "");
      }
   }

   public static void delete() {
      FileService.deleteFiles(logName);
   }
}
```

watermarking/src/main/java/utilities/TImeService.java

```
package utilities;

import java.util.concurrent.TimeUnit;

public class TimeService {

   private long startTime;
   private long endTime;

   public TimeService() {
      startTime = System.nanoTime();
   }

   public void stop() {
      endTime = System.nanoTime();
   }

   public Long getTime() {
      return TimeUnit.MILLISECONDS.convert(endTime - startTime, TimeUnit.NANOSECONDS);
   }
}
```

watermarking/src/main/java/utilities/WatermarkService.java

```java
package utilities;

import java.math.BigDecimal;
import java.math.RoundingMode;
import java.util.Collections;
import java.util.LinkedList;
import java.util.List;
import java.util.Random;

import entities.Fragment;
import entities.Measurement;
import entities.Range;
import entities.Request;
import entities.UsabilityConstraint;

public class WatermarkService {

   public static BigDecimal[] generateWatermark(Request request, UsabilityConstraint usabilityConstraint,
         Fragment fragment) {
      BigDecimal[] watermark = new BigDecimal[fragment.getMeasurements().size()];
      Random random = new Random(fragment.getSecretKey());
      random.setSeed(Long.valueOf(request.getNumberOfWatermark() + "" + Math.abs(random.nextInt())));

      // compute range probabilities
      List<BigDecimal> probabilities = new LinkedList<>();
      for (int i = 0; i < usabilityConstraint.getNumberOfRanges() / 2; i++) {
         BigDecimal probability;
         if (i > 0) {
            probability = BigDecimal.valueOf(0.5)
                  .divide(BigDecimal.valueOf(2).pow((usabilityConstraint.getNumberOfRanges() / 2) - i));
         } else {
            probability = BigDecimal.valueOf(0.5)
                  .divide(BigDecimal.valueOf(2).pow((usabilityConstraint.getNumberOfRanges() / 2) - (i +
                  1)));
         }
         probabilities.add(probability);
      }

      Collections.sort(fragment.getMeasurements());
      for (int i = 0; i < fragment.getMeasurements().size(); i++) {

         Measurement measurement = fragment.getMeasurements().get(i);
         Measurement prevMeasurement = new Measurement();
         Measurement nextMeasurement = new Measurement();

         // set previous measurement
         if (i > 0) {
            Measurement temp = fragment.getMeasurements().get(i - 1);
            prevMeasurement = new Measurement(temp.getDeviceId(), temp.getType(), temp.getUnit(),
               temp.getTime(), temp.getValue().add(watermark[i - 1]));
         } else {
            prevMeasurement.setValue(measurement.getValue());
         }

         // set next measurement
         if (i + 1 < fragment.getMeasurements().size()) {
            nextMeasurement = fragment.getMeasurements().get(i + 1);
         } else {
            nextMeasurement.setValue(measurement.getValue());
         }

         BigDecimal lowerBound = measurement.getValue().subtract(usabilityConstraint.getMaximumError());
         BigDecimal upperBound = measurement.getValue().add(usabilityConstraint.getMaximumError());

         // check valid bounds
         if (lowerBound.compareTo(usabilityConstraint.getMinimumValue()) < 0) {
            lowerBound = usabilityConstraint.getMinimumValue();
         }
         if (upperBound.compareTo(usabilityConstraint.getMaximumValue()) > 0) {
            upperBound = usabilityConstraint.getMaximumValue();
         }

         // compute boundaries for each case
         // v(t-1) < v(t) < v(t+1)
         if (measurement.getValue().compareTo(prevMeasurement.getValue()) > 0
               && measurement.getValue().compareTo(nextMeasurement.getValue()) < 0) {
```

```
            if (lowerBound.compareTo(prevMeasurement.getValue()) < 0) {
                lowerBound = prevMeasurement.getValue();
            }
            if (upperBound.compareTo(nextMeasurement.getValue()) > 0) {
                upperBound = nextMeasurement.getValue();
            }
        }
        // v(t-1) > v(t) > v(t+1)
        else if (measurement.getValue().compareTo(prevMeasurement.getValue()) < 0
                && measurement.getValue().compareTo(nextMeasurement.getValue()) > 0) {

            if (lowerBound.compareTo(nextMeasurement.getValue()) < 0) {
                lowerBound = nextMeasurement.getValue();
            }
            if (upperBound.compareTo(prevMeasurement.getValue()) > 0) {
                upperBound = prevMeasurement.getValue();
            }
        }
        // v(t-1) < v(t) > v(t+1)
        else if (measurement.getValue().compareTo(prevMeasurement.getValue()) > 0
                && measurement.getValue().compareTo(nextMeasurement.getValue()) > 0) {

            if (lowerBound.compareTo(prevMeasurement.getValue()) < 0) {
                lowerBound = prevMeasurement.getValue();
            }
            if (lowerBound.compareTo(nextMeasurement.getValue()) < 0) {
                lowerBound = nextMeasurement.getValue();
            }
        }
        // v(t-1) > v(t) < v(t+1)
        else if (measurement.getValue().compareTo(prevMeasurement.getValue()) < 0
                && measurement.getValue().compareTo(nextMeasurement.getValue()) < 0) {

            if (upperBound.compareTo(prevMeasurement.getValue()) > 0) {
                upperBound = prevMeasurement.getValue();
            }
            if (upperBound.compareTo(nextMeasurement.getValue()) > 0) {
                upperBound = nextMeasurement.getValue();
            }
        }
        // v(t-1) = v(t) < v(t+1)
        else if (measurement.getValue().compareTo(prevMeasurement.getValue()) == 0
                && measurement.getValue().compareTo(nextMeasurement.getValue()) < 0) {

            if (upperBound.compareTo(nextMeasurement.getValue()) > 0) {
                upperBound = nextMeasurement.getValue();
            }
        }
        // v(t-1) = v(t) > v(t+1)
        else if (measurement.getValue().compareTo(prevMeasurement.getValue()) == 0
                && measurement.getValue().compareTo(nextMeasurement.getValue()) > 0) {

            if (lowerBound.compareTo(nextMeasurement.getValue()) < 0) {
                lowerBound = nextMeasurement.getValue();
            }
        }
        // v(t-1) < v(t) = v(t+1)
        else if (measurement.getValue().compareTo(prevMeasurement.getValue()) > 0
                && measurement.getValue().compareTo(nextMeasurement.getValue()) == 0) {

            if (lowerBound.compareTo(prevMeasurement.getValue()) < 0) {
                lowerBound = prevMeasurement.getValue();
            }
        }
        // v(t-1) > v(t) = v(t+1)
        else if (measurement.getValue().compareTo(prevMeasurement.getValue()) < 0
                && measurement.getValue().compareTo(nextMeasurement.getValue()) == 0) {

            if (upperBound.compareTo(prevMeasurement.getValue()) > 0) {
                upperBound = prevMeasurement.getValue();
            }
        }
        // v(t-1) = v(t) = v(t+1)
        else if (measurement.getValue().compareTo(prevMeasurement.getValue()) == 0
                && measurement.getValue().compareTo(nextMeasurement.getValue()) == 0) {
            // do nothing
            }

            // compute ranges and range sizes
            List<Range<BigDecimal>> ranges = new LinkedList<Range<BigDecimal>>();

            BigDecimal lowerRange = measurement.getValue().subtract(lowerBound);
            BigDecimal upperRange = upperBound.subtract(measurement.getValue());

            BigDecimal lowerRangeSize = lowerRange
               .divide(BigDecimal.valueOf(usabilityConstraint.getNumberOfRanges() / 2), 15,
               RoundingMode.HALF_UP);
            BigDecimal upperRangeSize = upperRange
               .divide(BigDecimal.valueOf(usabilityConstraint.getNumberOfRanges() / 2), 15,
               RoundingMode.HALF_UP);

            // compute final ranges and assign probabilities
            for (int j = 0; j < (usabilityConstraint.getNumberOfRanges() / 2); j++) {

               BigDecimal lowerRangeMinimum =
                  lowerRange.negate().add(lowerRangeSize.multiply(BigDecimal.valueOf(j)));
               BigDecimal lowerRangeMaximum = lowerRange.negate()
                  .add(lowerRangeSize.multiply(BigDecimal.valueOf(j + 1)));

               BigDecimal upperRangeMinimum = upperRange.subtract(upperRangeSize.multiply(BigDecimal.valueOf(j +
                  1)));
               BigDecimal upperRangeMaximum =
                  upperRange.subtract(upperRangeSize.multiply(BigDecimal.valueOf(j)));

               if (lowerRange.compareTo(BigDecimal.valueOf(0.0)) == 0) {
                  ranges.add(new Range<BigDecimal>(upperRangeMinimum, upperRangeMaximum,
                        probabilities.get(j).add(probabilities.get(j))));
               } else if (upperRange.compareTo(BigDecimal.valueOf(0.0)) == 0) {
                  ranges.add(new Range<BigDecimal>(lowerRangeMinimum, lowerRangeMaximum,
                        probabilities.get(j).add(probabilities.get(j))));
               } else {
                  ranges.add(new Range<BigDecimal>(lowerRangeMinimum, lowerRangeMaximum, probabilities.get(j)));
                  ranges.add(new Range<BigDecimal>(upperRangeMinimum, upperRangeMaximum, probabilities.get(j)));
               }

            }
            Collections.sort(ranges);

            // select range
            Range<BigDecimal> selectedRange = new Range<BigDecimal>(BigDecimal.valueOf(0),
               BigDecimal.valueOf(0));
            BigDecimal randomNumber4RangeSelection = BigDecimal.valueOf(random.nextDouble());
            BigDecimal currentValue = BigDecimal.valueOf(0.0);
            for (int j = 0; j < ranges.size(); j++) {
               Range<BigDecimal> range = ranges.get(j);
               currentValue = currentValue.add(range.getValue());
               if (randomNumber4RangeSelection.compareTo(currentValue) <= 0) {
                  selectedRange = range;
                  break;
               }
            }

            // select mark
            BigDecimal randomNumber4ValueSelection = BigDecimal.valueOf(random.nextDouble());
            watermark[i] = selectedRange.getMinimum()
                  .add((selectedRange.getMaximum().subtract(selectedRange.getMinimum()))
                        .multiply(randomNumber4ValueSelection));
            watermark[i] = watermark[i].setScale(15, RoundingMode.HALF_UP);
         }
         return watermark;
      }
   }
```