Practical Privacy Preserving k-Nearest Neighbour in Outsourced Environments

I.S. Kroskinski

TU Delft

Practical Privacy Preserving k-Nearest Neighbour in Outsourced Environments

by

I.S. Kroskinski

to obtain the degree of Master of Science
at the Delft University of Technology,
to be defended publicly on Thursday August 24, 2023 at 14:00.

Student number: 4684958
Project duration: December 1, 2022 – August 24, 2023
Thesis committee:

Dr. Z. Erkin,	TU Delft, supervisor
Dr. A. Panichella.	TU Delft, Thesis Committee
J. Koehoorn,	Blueriq, daily supervisor

Cover image is retrieved from [69]

An electronic version of this thesis is available at http://repository.tudelft.nl/.

TU Delft

Preface

After six years of studying, my life studying Computer Science at the TU Delft comes to an end. From disliking cryptology, to developing an appreciation for its intricacies, simplicity, and recognizing its importance in society. Especially the course Privacy Enhancing Technologies showed me the importance and challenges of privacy, providing a basis for this work.

Throughout my 9 months of work on this thesis, I have received tremendous support and motivation from every one at the cybersecurity department. Special thanks to my supervisor Dr. Z. Erkin for pushing me and supporting me in hard times. Besides that, I want to thank Jelle Vos for sparking my interest in Fully Homomorphic Encryption and Florine Dekkers for providing value feedback. Also, I want to express my gratitude to Andrei and Jorrit for the fun conversations and coffee breaks.

This thesis would not have been possible without the people at the company Blueriq, with which I collaborated. I want to give special thanks to my supervisor at the company, Joost, for the interesting discussions, and Rick and Yuri, for providing me with all the resources I required.

Next, I want to thank all my friends I made during my time studying in Delft. Without Boris, Armin, Bram, Gerben and Robin I would not have been able to push myself to my limit.

Lastly, my family and friends in Haarlem have always supported me, and I could not be more grateful for the understanding and support I received.

I.S. Kroskinski
August 2023

Abstract

Organizations use cloud service providers for outsourcing their data, since this includes advantages such as: scalability, security and no need for in house experts. Therefore, outsourcing data to cloud providers results in reduced costs. The disadvantage of outsourcing data to a cloud provider, is that organizations are not in control of their own data. When organizations are not in control of their data, they are subject to privacy risks. Privacy risks should be avoided, especially when sensitive data such as medical or financial records are involved. Therefore, organizations protect their data by only outsourcing encrypted data to cloud providers. However, data analysis on encrypted data is significantly reduced due to computational and communicational overhead.

A commonly used data analysis method, such as k-Nearest Neighbour Search (k- NNS), is useful when for finding similar records in a database for a given query. Previous research shows success using k-NNS methods while preserving privacy, by using fully homomorphic encryption. However, previous solutions required the client to be online and help in the protocol, or make use of non-colluding servers.

Therefore, we introduce our k-NNS protocol, which outsources all the work to the cloud server and the client is not involved in the computation. Our k-NNS protocol shows success on data sets used to test k-NNS applications, however is significantly slower than solutions which involve the client or non-colluding servers.

Contents

List of Figures

List of Tables

List of Algorithms

1 Introduction

Our daily lives are increasingly dependent on the digital world, as technology has become more readily available [61]. With 4.6 billion active internet users worldwide [4] society is highly connected and technology, such as the internet, has become an integral part of our lives. Where 80% of the EU's population between 17–74 years make use of the internet on a daily basis [71]. From social media platforms such as Facebook, Instagram and Tiktok, to interacting with your local government through their website. Besides that, e-commerce sales worldwide is estimated to grow to $7,391 billion in 2025, where it was $4.938 billion in 2021 [28].

With widespread usage of new technology and the internet, our ability to collect and analyse data has improved. The abundance of data is clear, since in 2020 a total of 64.2 zettabytes were generated worldwide and is expected to grow to 180 zettabytes by 2025 [57]. Humans generate data when scrolling on social media, streaming services, online transactions and interacting with organizations in the public sector, among others. However, not only humans generate data any more but also machines, such as Internet of Things devices. The collection of all data generated by machines and humans is typically referred to as Big Data. But what makes it worth collecting and storing all this data?

1.1. Data Driven Society

In today's world, data is being used in various domains such as healthcare, finance, education, advertisement and transportation [40] [24]. In the past, decision-making was often done by human experts, based on intuition and experience, but with the advent of big data, this is no longer required. Instead, organizations gather and analyse large volumes of data to make calculated decisions. By analysing data, organizations can gain insights into customer behaviour, market trends, and other key factors that can impact their business. This is called a data driven society, where businesses, governments, and individuals rely heavily on data to make decisions [56]. Making decisions based on data analysis is called data-driven decision-making. Data driven decision-making has become increasingly popular due to the availability of vast amounts of data that can be analysed to extract valuable insights [12]. The use of data in decision-making helps organizations make informed decisions that are backed by evidence rather than guesswork.

There are two main advantages why organizations use data-driven decision-making. Firstly, it allows organizations to make decisions based on evidence found in the data, resulting in accurate and explainable decisions [67]. While in the past, decisions were based on intuition and experiences from experts. Secondly, data driven decision-making also allows organizations to work more efficiently [7], since computers are able to parse more information than a human.

Examples of success stories of data-driven decision-making are targeted advertisements and loan applications. Using data about individuals, advertisers are able to reach their target group efficiently, resulting in more profit. Because targeted advertisement is a powerful tool, companies exploit this and offer services to help organizations market their products or services. A known example of a company which utilizes data to provide targeted advertisement is Google. Just Google is able to generate $224.4 billion from advertisements in 2022, which has almost doubled since 2019 when it was $134.81 billion [9]. Another example of data driven decision-making are banks offering loans and insurances to con-

sumers. Using data of previous customers, predictions about new applications can be used to increase the organization's processes. The big data analytics market in the banking industry is estimated to be around $62.10 billion by 2025 [46]. There also exists non-profit use cases, such as customized treatment plans in healthcare applications and requests to the government. The healthcare sector has been making progress in adapting data driven decision-making, applying it to healthcare records [62]. Using data driven decision-making, hospitals are able to provide efficient and accurate healthcare treatment plans, as well as offloading workload from the employees. In 2025, it is expected that the market worth of big data analytics in healthcare would be worth $67.82 billion [59].

1.2. Cloud Services

As more data is generated from individuals, the task of storing and processing the data becomes challenging. Leading to problems for organizations who do not possess the capacity to store and process this data. There are companies, such as Amazon and Microsoft, who exploit this gap in the market and offer services such as storage and computing services. It is clear that organizations make use cloud service providers, since the cloud service provider industry it has grown by 23% in 2022 [65].

Cloud providers offer services such as on Data Analytics as a Service (DAaaS), which allows performing data analytics in an outsourced environment. DAaaS is a relevant topic since in 2019, because almost half of all data analytics were outsourced, a figure that is expected to grow [6].

Making use of DAaaS has four main advantages for organizations. One of the main advantages is that it is trivial for clients of cloud providers to scale up or down storage or computing as the demand rises or declines, respectively [20]. Not only is it trivial to scale up or down a service, deploying it on a cloud provider is often faster than purchasing hardware themselves and setting it up. Organizations therefore require no need for in-house experts to handle their own servers if they did wish to make use of a cloud provider. Having in-house experts is especially a problem when servers need to be set up across the globe. Whereas, by making use of a cloud provider, this is a trivial task.

Secondly, organizations themselves are not responsible for the security of the servers [20]. This includes the physical security of the actual servers, as well as cybersecurity. Cloud providers have hired security experts, who are able to deploy the most advanced firewall solutions. Because cloud providers have in-house expertise, cloud service providers can provider better protection against malicious internet traffic, That is not considering the often limited access to the physical location of the servers.

The next advantage includes reliable uptime and recovery options, since cloud providers often have multiple servers across the world [20]. In case of emergencies, cloud service providers are able to keep the service running and also provide recovery features.

The final advantages of using a DAaaS, is that it ultimately saves costs and human resources. Because clients of cloud service providers only pay for what they actually use [20], as well as not having to pay for replacing of hardware and hiring experts. Finally, because cloud providers have multiple clients, they are able to achieve economy of scale. Thus, cloud providers are able to offer a service cheaper than if clients were to use their own solution [47].

1.3. k-Nearest Neighbour Search

Cloud service providers makes it possible to outsource data driven decision-making, which is not a simple process and requires multiple steps to achieve [53], such as gathering, cleaning and analysing the data. This research will focus on the actual analysis step in the process of data driven decision-making. The analysis step performs an algorithm on a dataset and returns a result, from which a conclusion can be derived.

Data driven decision-making consists of four main analytic models: descriptive, diagnostic, predictive and prescriptive [35]. This thesis focuses on k-Nearest Neighbour Search (k-NNS), a form of predictive analytics, which aims to predict what is most likely to happen in the future based on previous events.

k-NNS is a form of a proximity search and can be defined as follows. For a query $\mathbf{q} \in \mathbb{R}^d$ and a database $\mathbf{D}$ with n records $\mathbf{x} \in \mathbb{R}^d$, where n is the number of records and d the number of attributes, find the k closest points for $\mathbf{q}$ in $\mathbf{D}$. Where the closeness is defined by a distance function, such as Euclidean or Manhattan distance. k-NNS returns either the entire point, or the ID of the k closest points to the query.

k-NNS is used in modern data analysis applications, where first data (images, text, audio, etc…) is converted to a feature vector. A feature vector is a representation of some data in the form of $v \in \mathbb{R}^d$, where d is the number of attributes. Some common applications where k-NNS is utilized are:

- **Information retrieval**: The k-NNS algorithm can be used to find the most similar documents or articles based on their content. By representing documents as a feature vector, k-NNS can retrieve relevant documents based on their similarity to a query.
- **Image recognition**: k-NNS can be employed for tasks such as image classification and object recognition. Given a new image, k-NNS can find the k most similar images from a database, allowing for image matching and categorization.
- **Recommendation systems**: k-NNS is used to suggest items or products based on the preferences or behaviour of similar users. By finding the nearest neighbours of a user, the system can recommend items that similar users have shown interest in.
- **Anomaly detection**: k-NNS can be applied to identify anomalies or outliers in datasets. By measuring the distance to the k nearest neighbours of a data point, it is possible to determine if it deviates significantly from its neighbours, indicating it as a potential anomaly.

1.4. Privacy & Ethical Challenges

Making use of cloud providers also has risks associated with them. One of the risks associated with the use of a cloud provider, is that cloud providers might not always be an honest party [74]. Cloud providers might read and use the data stored by clients for their own benefits. This is not limited by selling data to third parties and training their own models. Or even sell information to another organization making use of the same service [74].

Leaking sensitive information has serious consequences for the client of the cloud service provider, as well as for the provider itself. One well known occurrence of sensitive data being leaked, happened in 2019 when Cambridge Analytica gathered data from users on Facebook [66]. The data gathered, likely included information about the users “public profile, page likes, birthday and current city” [22]. Using this data, Cambridge Analytica was able to create a model about a user’s political alignment [60], which may have impacted individuals involved by influencing their votes in elections [36].

To avoid leaking sensitive information, the European Union introduced the General Data Protection Regulation (GDPR) [29] to protect data for European citizens. The main takeaway from the GDPR, is that organizations are required to use privacy-by-design. Privacy-by-design is to take privacy in account, throughout all stages of a design process. To outsource k-NNS to cloud service providers, the data should be protected, such that no sensitive information is leaked, but still needs to be able to get meaningful results from the data.

1.5. Privacy Preserving k-NNS

Techniques which help to protect sensitive data, while still allowing to perform operations on the data to gain meaningful results, are called privacy enhancing techniques. Privacy enhancing techniques have been used in combination with k-NNS, which we will refer to as Privacy Preserving k-NNS. Examples of privacy preserving techniques are anonymization, compression and encryption.

Anonymization techniques make changes to the data in order to not being able to identify an individual or a group of individuals, while still being able to make accurate approximate decisions. This is done in literature through k-anonymity [68], l-diversity [51] and t-closeness [48]. However, using anonymization techniques decreases accuracy of data mining techniques and cannot guarantee any privacy leaks.

Compressions techniques such as hashing are also common among literature. Hashing is used in bloom filters and locality sensitive hashing, which compress each individual record in the data into a vector. These techniques suffer from the same problems as anonymization techniques, such as accuracy loss and in the case of bloom filters, is insecure.

Lastly, encryption techniques convert the original data, otherwise known as plaintext, into unreadable data known as a ciphertext. Encryption techniques are able to not leak any information about the plaintext, by studying the ciphertext. The downside of applying an encryption scheme, is that it requires

computation to encrypt and decrypt, as well as limiting the ability to perform data analysis. One of the reasons organizations are hesitant to encrypt data before outsourcing, is because it increases computation and communication, as well as decreasing utilization. Just 11% of organizations have encrypted between 81-100% of the sensitive data they store in the cloud [65]. However, encryption is guaranteed to ensure privacy and accuracy.

With developments in privacy enhancing technologies, challenges in privacy preserving k-NNS are being overcome. However, current literature in privacy preserving k-NNS or similar problems (k-means) have tradeoffs in their implementation. Chen et al. [14] introduced a secure k-NNS algorithm which allowed users to query on a database with millions of records. However, they required that the database is stored in plaintext and that the client is involved in the computation. Which requires the client to still have computational and communicational capabilities. Works of [43] [23] [42], have tried to resolve these issues. But, they all require two or more non-colluding servers, which requires strong guarantees and trust between the parties. Finally, Shaul et al. [63] provide a probabilistic solution using standalone server, for k-ish classifier. Having these tradeoffs limits the utility of outsourcing k-NNS to cloud service providers.

1.6. Research Question

The aim of this research is to perform privacy preserving k-Nearest Neighbour, which enables organizations to outsource their data to cloud service providers without compromising on privacy, utility, accuracy. To achieve this, we focus on a k-NNS problem where no sensitive information is leaked. Besides that, the client should not be involved in the computation and thus should have minimal communication overhead. Finally, k-NNS should be able to run on a single stand-alone server. To achieve these requirements, this thesis will answer the following research question:

Is it possible to perform deterministic Privacy Preserving k-Nearest Neighbour search in outsourced environments using one cloud server and no interaction?

We define the following sub-questions, to help answer the main research question:

1. How can we guarantee the privacy of the data and the query in outsourced environments, while still maintaining deterministic results?
2. How can we construct k-NNS to use only one cloud server?
3. How can we construct k-NNS such that the client is not involved in the calculation?
4. How can we make use of the resources available in outsourced environments?

1.7. Contributions

We propose a Privacy Preserving k-NNS protocol which is able to protect the data, while maximizing utility. Utility is maximized since it allows deterministic queries on the data, without leaking sensitive information. Also, our solution does not require the assumptions of two or more non-colluding servers. Finally, the client is not involved in the computation, which saves resources for the client.

Our solution makes use of new advancements in Fully Homomorphic Encryption techniques, which makes it possible to perform non-linear operations in under a second. Limiting the number of bootstrapping operations in fully homomorphic operations, by means of limiting the plaintext domain, enables our solution to keep operations fast. Besides that, the protocol supports parallelization, which makes it suitable for outsourced environments.

This work aims to provide value for any individuals or organizations that require k-NNS in outsourced environments, as well as providing value for the cloud providers themselves. Firstly, individuals or organizations often do not outsource their data, since it is a tradeoff between privacy and utility. Performing k-NNS on plaintext data is fast and accurate, but unsecure. Performing Privacy Preserving k-NNS has reduced accuracy or has a significant computation or communication overhead. This paper aims to reduce the gap between usability and computation overhead for sensitive data. This is of interest for organizations that work with sensitive data that needs to be outsourced, such as hospitals, banks and the government.

Secondly, cloud providers would also benefit from this research. As organizations are more keen to outsource their data, cloud providers will receive more clients. Finally, this would not only have influence on cloud providers and their client, but also on the person whose data they store and use. Organizations would be more inclined to store data in ciphertext, since the drawbacks of doing privacy preserving k-NNS is dwindling.

1.8. Outline

The remainder of this thesis is organized as follows: Chapter 2 discusses the preliminaries of this thesis, which addresses all required knowledge. Related work done by previous researchers is discussed in Chapter 3. Chapter 4 discusses our Privacy Preserving k-NNS, including design goals, design choices and algorithms. Analysis of our protocol is done in Chapter 5, which includes a complexity analysis, security analysis and practical analysis using data sets used in previous research. Finally, Chapter 6 provides an overview of the discussion and any future work.

2

Preliminaries

This chapter will discuss knowledge and techniques used in privacy preserving k-NN, which our contribution will build on. We will briefly discuss privacy preserving k-NNS, different outsourcing scenarios, adversarial behaviours and homomorphic encryption.

2.1. Privacy Preserving k-Nearest Neighbour Search

k-Nearest Neighbour Search (k-NNS) is a technique used to analyse data to locate the k closest points in a given dataset, respective to a given query. An example of a k-NNS task is shown in Figure 2.1. k-NNS returns either the entire data points or just the ID of the top k points. Other derivatives of k-Nearest Neighbour algorithms, such as k-Means, can return a classification as a result.

In k-NNS, the dataset is represented by a set of data points, each having multiple attributes or features. These features could represent various characteristics of the data points, such as numerical measurements, categorical labels, or even complex structures. In modern data analysis applications, data such as, images, text, audio, etc…, is first converted to a feature vector. A feature vector is a representation of some data in the form of $v \in \mathbb{R}^d$, where d is the number of attributes.

The k-NN search process involves the following steps:

1. **Distance Metric Selection**: To measure the similarity between data points and a query, a distance metric needs to be chosen. The specific distance metric is dependent on the nature of the data and the problem. Examples of distance metric are: Euclidean, Manhattan and cosine distance.

2. **Distance Calculation**: When a query is received, the first step is to calculate the distance for each data point, respective to a given query. For each data point, the distance is defined as $\sum_{i=1}^{d} d(q_i, x_i)$, where d is the number of attributes and $d(q_i, x_i)$ is the distance function between the query and a data point attribute.

3. **Neighbour Selection**: To retrieve the k most similar points, a trivial solution is first sorting the items and returning the first k points. Other selection algorithms use complex data structures to speed up finding the most similar points.

Factors influencing the effectiveness of k-NNS are dependent on the underlying data and the implementation. Factors include the number of rows and attributes in a data set, distance metric and underlying data structures. Other performance increases can be obtained by using approximations.

Privacy Preserving k-NNS is a specific instance of k-NNS, which limits the amount of knowledge a certain party can obtain. The client is not allowed to know anything beyond the specific query and the result of that query. The server should not be able to recover any sensitive information and should only participate for the computation capabilities. Any third-parties should not be able to known anything.

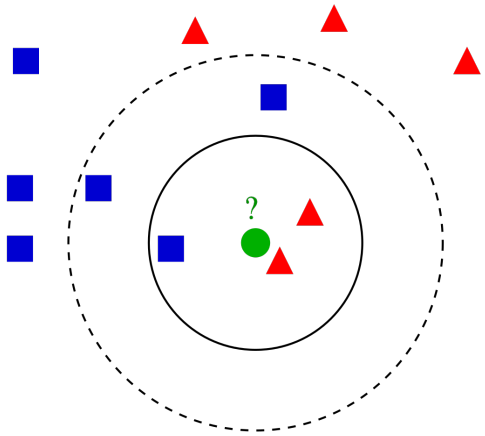


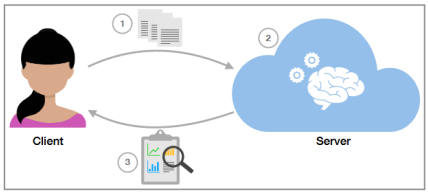
Figure 2.1: An example of a k-NNS task (Retrieved from: [3])

2.2. Outsourcing Scenarios

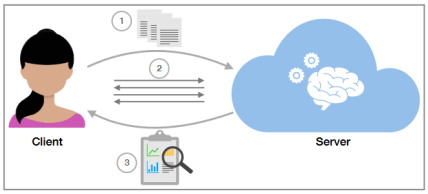
Outsourcing Privacy Preserving k-NN tasks can consist of different scenarios, based on how the client and server(s) interact with each other during computation. Different scenarios are employed based on the computation limitations and the tasks' adaptability. Three common scenarios which will be discussed: *Standalone Server*, *Client-Server* and a *Non-Colluding Servers* scenario. The client and server are defined as follows:

- *Client* refers to the party which outsources the computational tasks and performs a query.
- *Server* refers to the party or parties, which performs computational tasks for a client, such as a cloud service provider.

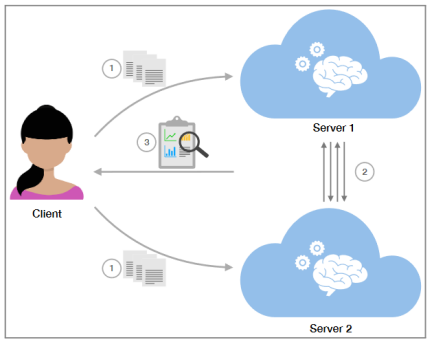
Figure 2.2 provides an overview of the configuration and interaction between the client and the server(s).



(a) Standalone server scenario



(b) Client-server scenario



(c) Non-colluding servers scenario

Figure 2.2: Commonly used scenarios when outsourcing computational tasks (Retrieved from: [70])

2.2.1. Standalone Server

In the standalone server scenario, as illustrated in Figure 2.2a, the client outsources the task to the server and is not involved in any computation. Because the client is not involved in any computation, there is also no further interaction required between the client and the server. Only the server is involved in the computation of the task and sends the result back to the client. The standalone server is the ideal scenario for outsourcing computation tasks, because the client does not require any computation

resources for itself. Since the standalone server scenario does not require any interaction, the client can be *offline* during the computation of the task.

2.2.2. Client-Server

The client-server scenario, as described in 2.2b, is similar to the standalone server scenario, but this time the client is involved in the computation. Because the client is involved in the computation, it is required that the client has computational and bandwidth resources. Also, now the client is required to be online during the computation of the task. The client-server scenario is used over the standalone server scenario to speed up computations or perform operations that are not possible without disclosing sensitive information.

2.2.3. Non-Colluding Servers

When there are not enough computational resources on only one cloud server, as in the standalone server scenario, but the client cannot be involved in the computation as described in the client-server scenario, the non-colluding servers scenario can be used. The non-colluding servers scenario, illustrated in Figure 2.2c, uses two or more servers who do not collude with each other, such that the multiple servers work together to receive the result. It is crucial that the servers do not collude with each other, otherwise the servers would be able to retrieve private information. Non-Colluding servers scenario therefore requires strong guarantees between the parties and trust.

2.3. Adversarial Capabilities & Behaviour

To design a privacy preserving protocol, it is necessary to define the capabilities and behaviours of any adversaries. An adversary can be the client, the server(s) or third-parties. First, we will discuss different capabilities of adversaries as described in literature. Secondly, we will describe three adversarial behaviours which are used in literature.

2.3.1. Adversarial Capabilities

To prove that a protocol is secure against adversaries, it is necessary to understand how powerful the adversary is. The literature differentiates between bounded and unbounded adversaries. We briefly discuss the definition of bounded and unbounded adversaries.

Bounded Adversaries have limited computational capabilities to break a cryptographic scheme. Breaking a cryptographic scheme means it is possible to retrieve the plaintext with a probability of 1. If a crypto scheme is secure against a bounded adversary, the crypto scheme is called computationally secure. Computationally secure schemes require the adversaries to perform N operations using the best known algorithm, where N is large, such that it is considered infeasible.

The number of operations to break the crypto scheme can be expressed as 2^n, where n is the bit security level. The minimum n-bit security today is defined to be $n = 80$, as recommended by NIST [8]. The number of security bits recommended, compensating for future advancements in computing power, is defined to be $n = 128$.

Unbounded Adversaries have unlimited computation capabilities and thus requires a stricter definition for a secure crypto scheme. A cryptographic scheme which is secure against unbounded adversaries, is called unconditional or information theoretic secure. A crypto scheme which is unconditionally secure, requires that it does not leak any information, which is defined as perfect secrecy. A formal definition of perfect secrecy is defined in Definition 1. An example of a cryptographic scheme which provides perfect secrecy is One Time Pad [52].

Definition 1 (Perfect Secrecy)**.** *A crypto scheme provides perfect secrecy when:* $p(P = m|C = c) = p(P = m)$*, for all plaintexts* m *and all ciphertexts* c*.*

2.3.2. Adversarial Behaviour

Besides defining adversarial capabilities, adversarial behaviours should also be considered when analysing a protocol. In this section, we briefly discuss the different adversarial behaviours used in literature, such as semi-honest, malicious and covert.

Semi-Honest or honest-but-curious adversarial behaviour is the weakest behaviour, but used in situations when parties have a mutual trust. In the semi-honest behaviour, the adversary does not deviate from the protocol, however it tries to learn as much as possible during the protocol. The information can be obtained from inputs, outputs and intermediate messages and calculations.

Malicious adversarial behaviour allows the adversary to deviate from the protocol to learn as much information as possible. The malicious model is a stronger behaviour model than a semi-honest model and is therefore desired. However, to make a protocol secure against a malicious adversary has an overhead compared to a semi-honest model.

Covert adversarial behaviour allows an adversary to behave malicious and to learn as much information as possible, but with a chance of being detected. This adversarial behaviour model is a compromise between the semi-honest and malicious behaviour model.

2.4. Approach

In this section, we will briefly describe different privacy preserving techniques which can be used to perform Privacy Preserving k-NNS.

Anonymization can be used to perform Privacy Preserving k-NNS efficiently. Using anonymization techniques allows the protocol to perform the operations in plaintext, for which already efficient algorithms exist. Anonymization works in the standalone server scenario and can be highly parallelizable. However, before outsourcing the data to the server, the client first has to anonymize the dataset. Anonymizing a dataset is a computationally expensive task in itself and therefore requires the client to have computationally capabilities, which we are trying to outsource. Besides that, anonymization limits the amount of information an adversary is able to gather, but it cannot guarantee that no information is leaked.

Locality Sensitive Hashing (LSH) is also an efficient method of performing k-NNS efficiently. LSH can be performed in the standalone server scenario, be parallelizable, as well as achieving privacy. The disadvantage of using LSH, is that it is probabilistic. Since LSH is probabilistic, it is possible that a non-similar record is returned to the result.

Multi Party Computation (MPC) techniques, such as secret sharing, garbled circuit and using partial homomorphic encryption schemes in the non-colluding servers setting have been used in literature already. The advantages of using MPC, is that is able to protect the query, the result and the data using cryptographic primitives. MPC is preferred in scenarios where the client has computational capabilities or the server is allowed to communicate with other servers, since MPC is often faster than fully homomorphic encryption.

Fully Homomorphic Encryption (FHE), allows the server to be able to compute any arbitrary function in the standalone server scenario and these computations can be parallelized. Since FHE can compute any arbitrary function, the protocol can have the same behaviour as the plaintext implementation. However, the disadvantage of using FHE is the significant computational overhead for the server. Depending on the FHE scheme, either non-linear operations or linear operations are expensive.

For our protocol, we will be making use of Fully Homomorphic Encryption, although FHE has a significant computation overhead. FHE allows our protocol to be non-interactive without the use of non-colluding servers, while preserving the privacy and same behaviour as plaintext implementations.

2.5. Homomorphic Encryption

Homomorphic Encryption makes it possible to perform operations on ciphertexts, which transforms the underlying plaintext as well. Operations on ciphertexts are possible because of homomorphism, which is a structure preserving map between two algebraic structures. Some crypto schemes support homomorphic operations, such as additive and multiplicative homomorphism. Crypto schemes that support homomorphic operations are called homomorphic encryption schemes. Homomorphic encryption

schemes can be subdivided into multiple categories: partial, levelled and fully homomorphic.

Partial homomorphic encryption schemes only support one homomorphic operation, such as addition or multiplication. Partial homomorphic schemes are therefore called additive or multiplicative homomorphic, based on their supported homomorphic operation. An example of an additive homomorphic scheme is the Paillier scheme [54].

Levelled homomorphic encryption schemes support a limited number of homomorphic operations until the results become unintelligible. Levelled homomorphic encryption schemes can support both additive and multiplicative homomorphism, which allows computing an arbitrary small function.

Fully homomorphic encryption schemes allow unlimited number of additions and multiplications. Fully homomorphism was first introduced by Gentry [30], which introduced a method called bootstrapping, to convert a levelled homomorphic encryption scheme to a fully homomorphic encryption scheme. Using an unlimited amount of additions and multiplications, it is possible to compute any arbitrary function. However, although it is possible to compute any arbitrary function in theory, in practise this is infeasible since the bootstrapping operations are computationally expensive.

2.5.1. Fully Homomorphic Encryption Schemes

In the literature, numerous fully homomorphic encryption (FHE) schemes have been proposed, however not one FHE scheme performs best for all tasks. In order to perform non-linear operations using FHE, arithmetic circuits can be used. Arithmetic circuits work on a bitwise level, such that one is able to create logic gates. When performing operations with two integers, they first have to be encoded as binary, then multiple additions and multiplications are applied on those bits. Therefore, performing non-linear operations has a significant overhead and different FHE schemes use different methods to make operations efficient. We will briefly discuss various FHE schemes and their strong and weak points.

Two of the earlier FHE schemes are BGV [10] and BFV [27], for which homomorphic operations are slower than other works. However, BGV and BFV support Single Instruction Multiple Data (SIMD) [64] operations. SIMD operations makes it possible to pack multiple messages into one ciphertext, thus performing multiple operations in one instruction. Making use of SIMD operations make it possible to perform fast comparisons, as described in [33]. Therefore, for some arithmetic circuits it might be faster to use an FHE scheme that supports SIMD, but is slower in single operations.

Another FHE scheme which supports SIMD, is CKKS [16]. CKKS has the benefit of supporting complex and real numbers, while allowing to pack more bits into a ciphertext. However, CKKS is an approximate scheme and therefore cannot be used in applications which require high precision and is slower than alternatives.

TFHE [18] is an FHE scheme which does not support SIMD, but rather performs all operations in binary. By performing all operations on a bit wise level, it is trivial to implement non-linear operations. However, the gain in efficiency for non-linear operations is compensated in the lack of efficiency in performing additions and multiplications on integers. Performing additions and multiplications using TFHE requires implementing whole arithmetic circuits, which as an overhead. Another advantage of TFHE, is the performance of the bootstrapping operations. Besides that, TFHE support programmable bootstrapping, which allows one to perform a function during the bootstrapping operations, speeding up arithmetic circuits.

Finally, Klemsa and Onen [44] proposed the FHE scheme Parmesan, which builds on top of TFHE. Parmesan improves the efficiency of multiple operations, by making specific operations parallelizable and maximizing the number of cores used. But, not all operations are yet implemented. The authors claim to have a speed-up of 2.3 times when squaring and adding, on a 12 core machine. However, Parmesan made use of all the 12 cores while Concrete, a previous version of TFHE, used only one core. Therefore, even if Parmesan is faster, it would be better to parallelize each operation instead of a single operation.

To conclude, BGV and BFV are fast only in specific instances where SIMD can be used, and slower otherwise. The same goes for CKKS, which should only be used when complex or real numbers are required. Finally, TFHE is the best choice for performing an arbitrary circuit because of the flexibility and the fast bootstrapping. TFHE is preferred over Parmesan when the operations can be parallelized, and Parmesan only outperforms TFHE when all operations need to be performed sequentially on a multithreaded machine. Therefore, in our work we will consider the TFHE scheme, where the implementation of the scheme can be found here [76] and is provided by Zama.

2.5.2. TFHE

One specific instance of a fully homomorphic encryption scheme is TFHE [18]. TFHE supports additions and multiplications, as well as non-linear operations through an arithmetic circuit. The security of TFHE depends on the underlying assumption that the Learning With Error (LWE) [58] problem is hard to solve. Encryption schemes that depend on LWE are considered to be resistant against attacks from quantum computers.

Encrypting data using TFHE requires that the plaintext data must first be encoded in a unsigned integer. When encoding a value, as seen in Equation 2.1 and Figure 2.3, the value is set to the most significant bits of an unsigned integer and some noise is added to the least significant bits for security. After encoding, it is possible to encrypt the encoded message to a LWE ciphertext. We refer the reader for the specific implementation of encrypting using TFHE to the original paper [18].

$$m' = (m * \Delta) + \epsilon \tag{2.1}$$

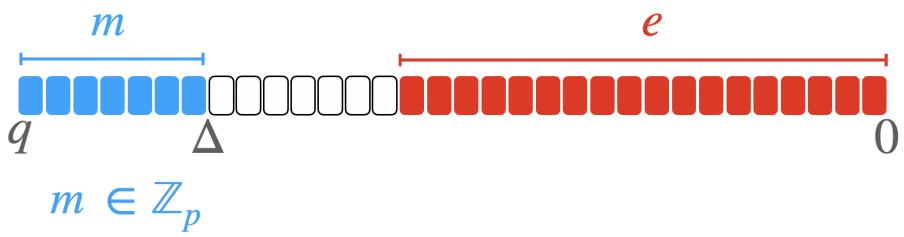


Figure 2.3: An encoding of a message m using TFHE (Retrieved from: [75])

Performing operations on TFHE ciphertext is possible up to a certain limit, otherwise the underlying message gets corrupted. This is due to the noise that is being introduced in the encoding of the message, as seen in Equation 2.1. Performing operations on the ciphertexts means not only are the operations performed on the underlying message, but also on the noise. The noise grows additively for additive operations and multiplicatively for multiplicative operations. In order to protect the underlying message from the error corrupting the message, a padding is used between the message and the noise, as seen in Figure 2.3.

To use more operations, the noise level needs to be reset and for which techniques by Gentry are [30] used, by performing a bootstrapping operations. TFHE can support up to a certain number of operations before bootstrapping is needed. The number of operations is depended on the operations applied and the parameters chosen for the cryptographic scheme. As described before, the noise grows multiplicatively with multiplications and therefore requires bootstrapping more often. Bootstrapping allows TFHE to perform unlimited multiplications and additions, however bootstrapping is computationally expensive.

Since bootstrapping is computationally expensive, TFHE support Programmable Bootstrapping (BPS) [17], allowing to perform univariate functions while bootstrapping. While not reducing the efficiency of bootstrapping, BPS does increase the utility of bootstrapping. Since TFHE supports evaluating a function using BPS, it also supports nonlinear operations, which makes arithmetic circuits significantly faster.

3

Related Work

There have been multiple studies about performing Privacy Preserving k-Nearest Neighbour protocols in outsourced scenarios. This section provides an overview of different techniques used to perform k-NN. Not only k-NNS are discussed, but also other variants of k-Nearest Neighbour, since the building blocks in these algorithms are similar.

First, in Section 3.1 we discuss how researchers perform k-NN on plaintext without leaking sensitive information using anonymization techniques. Secondly, in Section 3.2 compression techniques are discussed, which are probabilistic schemes that use efficient data structures to quickly find similar entries. Next, Section 3.3 describes different cryptographic solutions, where we focus on interactive and non-interactive protocols. Finally, we summarize all related work in Section 3.4.

3.1. Anonymization Techniques

Some researchers argue that encryption techniques are not necessary, since it is possible to achieve privacy by making changes in the data such that no group or individuals can be distinguished. Advocates of anonymization techniques reason that performing data driven decision-making on encrypted data is not usable in some situations, since it has a large computation and communicational overhead. Anonymization techniques allow data driven decision-making in plaintext, which has significant computational and communicational advantages, with minimal privacy risks. Anonymization techniques only introduce overhead during the initial setup, after which it introduces no additional complexity. Therefore, we briefly discuss commonly used techniques such as k-anonymity [68], l-diversity [51] and t-closeness [48].

To understand anonymization techniques, it is important to distinguish different types of attributes in a dataset. Attributes can be classified under three categories: identifier, quasi-identifier, sensitive and non-sensitive attributes. Identifiers are attributes which enable an adversary to fully identify an individual on solely one attribute. Quasi-identifiers are attributes which can somewhat identify an individual, but a combination of quasi-identifiers can be used to fully identify an individual with external knowledge. Sensitive attributes contain data that can lead to harm, such as identity theft or other crimes. The European Union [21] defines the following characteristics of data as sensitive:

- personal data revealing racial or ethnic origin, political opinions, religious or philosophical beliefs;
- trade-union membership;
- genetic data, biometric data processed solely to identify a human being;
- health-related data;
- data concerning a person's sex life or sexual orientation.

Non-sensitive attributes are data that is not sensitive. Anonymization techniques aim to protect sensitive data of individuals, by not being able to identify the individuals themselves.

One of the early techniques used to protect sensitive data by altering the data, is **k-anonymity** [68]. k-Anonymity aims to alter the data, such that each individual cannot be distinguished from at least

$k-1$ other individuals in the dataset, by performing operations on identifiers and quasi-identifiers. It is possible to achieve k-anonymity by performing suppression and generalization techniques on the data, such that there are no identifiers for every combination of quasi-identifiers, such that it can be indistinctly matched to at least k individuals [68]. Suppression techniques aim to remove values in attributes, whereas generalization replaces values by an approximation or range. Table 3.2 is a 4-anonymized table of Table 3.1.

k-Anonymity is a tradeoff between privacy and accuracy. Increasing k means more privacy, however this results in lower accuracy since more data needs to be changed. There are multiple permutations of the dataset possible to achieve k-anonymity. However, not all permutations result in good accuracy. Therefore, finding the optimal k-anonymity set has been an active area of research [41].

Further research has shown that k-anonymity by itself is not able to provide indistinguishability, since it is vulnerable to at least two attacks. First, an adversary is able to retrieve the sensitive attribute from an individual in an k-anonymized dataset, if the data is vulnerable to a homogeneity attack [51]. A homogeneity attack is possible when there is no diversity for sensitive attributes in the group generated by k-anonymity. As seen in Table 3.2, the third group only includes the sensitive attribute *Cancer*. If an adversary knows the victim lives on zip code 13053 and is 31 years old, this only leaves records 9, 10, 11 and 12. Since records 9 to 12 only have the sensitive attribute **Cancer**, the adversary can distinguish that the victim has a certain sensitive attribute.

Secondly, k-anonymity is vulnerable to background knowledge attacks [51]. Background knowledge attacks use external information to narrow down the number of possible sensitive attributes. Using the anonymized dataset in Table 3.2, the victim aged 21, living in Japan on zip code 13068, the adversary can conclude that the victim is in the first group (records 1-4). The adversary can now only conclude that the victim either has a heart disease or a viral infection. But, using the knowledge that Japanese people are unlikely to suffer from a heart disease, the adversary can conclude that the victim must suffer from a viral infection.

Table 3.1: Example of inpatient microdata (from [51])

	Non-Sensitive			Sensitive
	Zip Code	Age	Nationality	Condition
1	13053	28	Russian	Heart Disease
2	13068	29	American	Heart Disease
3	13068	21	Japanese	Viral Infection
4	13053	23	American	Viral Infection
5	14853	50	Indian	Cancer
6	14853	55	Russian	Heart Disease
7	14850	47	American	Viral Infection
8	14850	49	American	Viral Infection
9	13053	31	American	Cancer
10	13053	37	Indian	Cancer
11	13068	36	Japanese	Cancer
12	13068	35	American	Cancer

To protect against homogeneity and background knowledge attacks, **l-diversity** [51] was introduced, which is an extension to k-anonymity. To achieve l-diversity, first perform k-anonymity, after that the dataset requires that for each quasi-identifier block, there are at least l well represented sensitive values for sensitive attributes. A 3-diverse arrangement of the dataset of Table 3.2 is shown in Table 3.3.

l-Diversity protects against homogeneity attacks and limits the adversaries' ability to perform a background knowledge attack. By increasing l, the adversary requires more background knowledge to conclude a victim's sensitive attribute. However, increasing l can lead to inaccurate results, since more generalization needs to be performed to reach l-diversity. Besides that, it is not always possible to get l-diversity, since there may not be enough sensitive attributes.

Datasets conforming to l-diversity can still leak information about sensitive attributes if an adversary has background knowledge about the global distribution of the data [48]. l-Diversity leaks information because it is vulnerable against skewness and similarity attacks. Skewness attacks can happen when the overall distribution of the sensitive value is skewed. It allows an adversary to wrongly classify a large

Table 3.2: 4-anonymous inpatient microdata (from [51])

	Non-Sensitive			Sensitive
	Zip Code	Age	Nationality	Condition
1	130**	<30	*	Heart Disease
2	130**	<30	*	Heart Disease
3	130**	<30	*	Viral Infection
4	130**	<30	*	Viral Infection
5	1485*	≥40	*	Cancer
6	1485*	≥40	*	Heart Disease
7	1485*	≥40	*	Viral Infection
8	1485*	≥40	*	Viral Infection
9	130**	3*	*	Cancer
10	130**	3*	*	Cancer
11	130**	3*	*	Cancer
12	130**	3*	*	Cancer

portion of individuals with a sensitive attribute that can negatively affect the result. Similarity attacks happen because l-diversity does not differentiate between semantically similar sensitive values.

Table 3.3: 3-Diverse inpatient microdata (from [51])

	Non-Sensitive			Sensitive
	Zip Code	Age	Nationality	Condition
1	130**	≤40	*	Heart Disease
4	130**	≤40	*	Heart Disease
9	130**	≤40	*	Cancer
10	130**	≤40	*	Cancer
5	1485*	>40	*	Cancer
6	1485*	>40	*	Heart Disease
7	1485*	>40	*	Viral Infection
8	1485*	>40	*	Viral Infection
2	130**	≤40	*	Heart Disease
3	130**	≤40	*	Viral Infection
11	130**	≤40	*	Cancer
12	130**	≤40	*	Cancer

t-closeness [48] aims to solve the issues in l-diversity, by limiting the distance between sensitive data distributions to be within a certain limit t. A table conforms to t-closeness, when for each equivalence class "the distance between the distribution of a sensitive attribute in this class and the distribution of the attribute in the whole table is no more than a threshold t" [48]. The original paper used the Earth Mover Distance (EMD) as a metric to calculate the distance between two distributions. EMD can be described as the minimal amount of work needed to transform one distribution into another by moving the distribution mass. t-Closeness solves skewness attacks and limits the ability to perform similarity attacks. Therefore, reducing the information an adversary is able to obtain from the data, even with background knowledge.

Anonymization techniques make changes to the data, such that it is hard for an adversary to identify groups or individuals. Although, anonymization cannot guarantee any leakage from the dataset when the adversary has enough background knowledge. The advantage of anonymization is that one is able to perform complex operations on the plaintext. Because it is possible to perform operations on the plaintext, faster and more accurate algorithms can be deployed. However, since changes are made to the data, the results might differ from the original data.

3.2. Compression Techniques

Compression techniques use one-way functions to convert plaintext into a hash value. The advantage of compression techniques, is that you do not work on the plaintext directly, and it provides privacy by that it is hard to find the underlying plaintext from the hash value. Besides that, compression techniques are often faster than trivial techniques such as linear scan through all the data, and are able to achieve a sublinear time complexity. This section discusses two compression techniques that show promise in current research: bloom filters and locality sensitive hashing.

Vatsalan and Christen [72] make use of bloom filters to efficiently calculate similarity scores over a database and a query. Bloom filters are a probabilistic data structure which are known to be space-efficient. Bloom filters are a useful tool for checking whether an element is a member of a set. It is possible for bloom filters to return a false positive since it is a probabilistic scheme, but not false negatives.

A bloom filter consists of a m size array filled with all zeroes. Using k different hash functions which map a specific record or the attributes separately, into the m dimension array by setting a bit into that position in that array. Using this m dimension array, it is possible to check for similarity by repeating the hashing operation on any query to check if the query is in the set. If there is a bit set in the query bloom filter array, but not in the array of the data point, it is definitely not a member of the set. If all bits are set in the query array, then there is a high likelihood that the query is a member of the set.

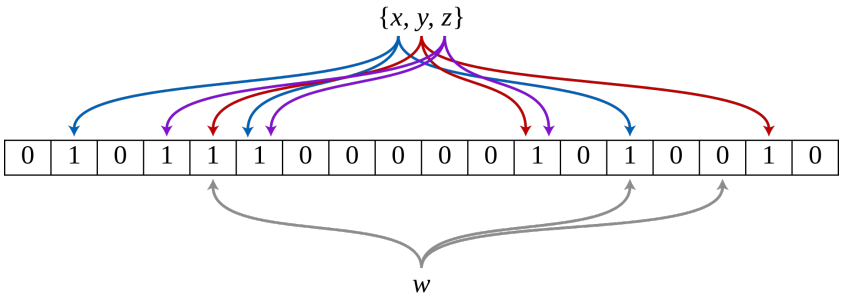


Figure 3.1: An example of a bloom filter where the query w is not a member of the set (Retrieved from: [26])

Similarity is based on the Dice coefficient similarity, as shown in Definition 2, where v_i is a bloom filter with x_i the number of bits set and c the number of common bits that are set to 1. The similarity score is a number between 0 and 1, where 0 means no similarity and 1 means they are the same. Although bloom filter are efficient structures for finding similar items, Christen et al. have proven that bloom filters are insecure [19].

Definition 2 (Dice Coefficient Similarity)**.** $sim_M(v_1, v_2) = \frac{2 \times c}{x_1 + x_2}$

Huang et al. [31] make use of locality sensitive hashing (LSH) to find similar records efficiently. LSH is a compression technique which uses multiple hashing functions to find similar items. Using multiple hashing functions, it is possible to map 'similar' items into the same hashing buckets. If items are in the same bucket, then there is a high probability that the items are similar. Picking a specific family of hashing functions $\mathcal{F}$, it is possible to determine the likelihood that items are similar when they are in the same bucket.

The hash function family $\mathcal{F}$ maps $h : M \rightarrow S$, where M is the data point and S is a collection of hash buckets. The hash function family $\mathcal{F}$ is locality sensitive when for the points p, q [34]:

- $(d(p,q) \leq R \rightarrow h(p) = h(q)) \geq P_1$ (Is in the bucket when they are similar).
- $(d(p,q) \geq cR \rightarrow h(p) = h(q)) \leq P_2$ (Is in the bucket when they are NOT similar).

A LSH family $\mathcal{F}$ is useful when $P_1 > P_2$, so when the likelihood of two items that are in the bucket are similar is bigger, than if they are not similar. However, all LSH techniques are probabilistic, therefore it is possible that some entries are not related at all.

3.3. Cryptographic-based k-NN

In this section, we discusses related work which makes use of cryptographic techniques to perform k-NN protocols. First, interactive protocols are discussed, which are protocols that work in the client-server and non-colluding servers scenario After that, protocols in the standalone server are considered.

3.3.1. Interactive Protocols

One of the first papers written to perform Privacy Preserving k-Nearest Neighbour (k-NNS) was done by Elmehdwi et al. [23]. Elmehdwi et al. provided the first protocol, which was secure, accurate and outsourced the majority of the computation to the cloud server. The protocol uses Euclidean distance as distance metric and made use of a partially homomorphic crypto scheme, the Paillier crypto scheme, which allows additive operations on ciphertexts. However, the Paillier crypto scheme does not support multiplication and non-linear operations, which are necessary to calculate the distances and finding the minimum distances without communication. To calculate Euclidean distances and finding minimum distances, one needs to be able to perform multiplication and comparisons. Therefore, Elmehdwi et al. provide interactive protocols to perform multiplications and finding minimum distances.

To calculate Euclidean distances, the authors use a Secure Multiplication protocol. Using two rounds of interaction, Elmehdwi et al. are able to multiply two encrypted numbers. The Secure Multiplication is then used to compute Secure Squared Euclidean Distances.

To compare two different values encrypted by the Paillier cryptosystem, the authors use bit decomposition, where each integer is represented as an encrypted binary representation. Then, after finding the first index where one vector of bits is smaller, that vector is reconstructed to an integer again.

The protocol is designed to be used in the non-colluding servers scenario, such that the client is not involved in any computation. The overhead for the protocol is $\mathcal{O}(n * (l + m + k * l * log_2 n))$, where n is the number of rows, l the number of bits to represent an integer after squaring, k the number of minimum values to find and m the number of attributes.

An extension on the work of Elmehdwi et al. is provided by Kim et al. [43], which aimed to improve the efficiency of the protocol. Kim et al. were able to realize an efficiency improvement in the protocol by making use of data structures, which are efficient in searching for points. The data structure used is a k-d tree, which is able to partition points in k dimensions. Because of these partitions, one is able to eliminate irrelevant portions of the k-d tree, finding the point with the minimum distance faster. One downside of using a k-d tree is, it is only efficient when $n >> 2^k$, where n is the number of records and k the number of attributes of the data. Also, the authors introduce an improved version of the Secure Squared Euclidean Distance protocol introduced by Elmehdwi et al. and propose the Enhanced Secure Squared Euclidean Distance protocol. The speed-up is obtained by making use of data packing, which puts multiple data into one ciphertext, allowing to perform multiple operations with only one operation. As a result of these performance increases, Kim et al. are able to gain a speed-up of up to 24 times in comparison with the work of Elmehdwi et al, although the authors do not provide a complexity analyses. However, to optimize the performance gains, it requires choosing the right parameters for the k-d tree.

Kesarwani et al. [42] also provide a Privacy Preserving k-NN protocol in the non-colluding scenario. However, the authors use a levelled homomorphic encryption scheme, Brakerski-Gentry-Vaikuntanathan (BGV), which allows a limited number of additions and multiplications, as well as support data packing. The author's choice of using a levelled homomorphic encryption scheme is based on the fact that it supports more operations than a partial homomorphic scheme, but is faster than a fully homomorphic encryption scheme. Using BGV, Kesarwani et al. are able to compute k-NN with minimal communication overhead, since the protocol only uses a single round. The number of homomorphic operations is $\mathcal{O}(n * (k + d + \mathcal{D}))$, where n is the number of records, k the number of minimum values to find, d the number of attributes and D the number of distance to calculate.

Zheng et al. [77] also make use of k-d trees in the non-colluding scenario, as well as introducing a scalar preserving crypto scheme, MASPE. MASPE is able to preserve the sign of the scalar, which makes it possible to perform cheap comparisons, which is especially useful for searching in k-d trees. Equation 3.1 & 3.2 shows the relation of the sign between the plaintext and the ciphertext for MASPE

and a query q, a query token TK_q, a record x and the ciphertext of x, C_x:

$$x \circ q \leq 0 \Leftrightarrow C_x \circ TK_q < 0. \tag{3.1}$$

$$x \circ q > 0 \Leftrightarrow C_x \circ TK_q > 0. \tag{3.2}$$

MASPE is an improved version of the ASPE crypto scheme [78], since ASPE was deemed to be insecure under known plaintext attacks [49]. Other variants of ASPE, such as MRSE [13] and MKFSE [73], were also proven to be insecure. So it is unsure if MASPE is secure.

A recent paper from Chen et al. [14] proposes a protocol for solving Secure k-NNS, which aims to protect the query of the client from the server and the data of the server for the client in the client-server scenario. Secure k-NNS differs from Privacy Preserving k-NNS, since in Secure k-NNS the server is allowed to have access to the plaintext of the sensitive data but keeping the query private, while in Privacy Preserving k-NNS the server is not allowed to have access to the sensitive data.

Although Secure k-NNS differs from Privacy Preserving k-NNS, Chen et al. do use useful primitives such as Garbled Circuits, Secret sharing, Homomorphic Encryption and Oblivious RAM. Using the aforementioned cryptographic primitives, the authors propose the SANNS protocol. SANNS is a Secure Approximate k-Nearest Neighbour Search protocol which supports large databases such as SIFT [39], which contains a million entries and 128 attributes. SANNS is able to support large databases, since it uses the strong points of each cryptographic primitive. Homomorphic encryption is used for distance computation, garbled circuits for top-k selection, Oblivious Ram for securely retrieving values and using secret sharing for connecting all the primitives.

3.3.2. Standalone Protocols

Shaul et al. [63] propose a k-ish NN classifier in the standalone scenario. Instead of returning exactly k records or IDs, the protocol returns the classification of the majority of the $k \simeq k$ results. To be more exact, $k/2 < \kappa < 3k/2$. Since k does not have to be exact, the authors are able to improve the efficiency of the protocol. The improvement in efficiency is necessary since the server has to do all the computation itself, and Shaul et al. make use of the levelled homomorphic BGV crypto scheme [10] which as a significant computational overhead.

The authors achieve the efficiency improvement by assuming the underlying data follows a Gaussian distribution. Next, Shaul et al. calculate μ and σ on the underlying data and sampling multiple times from this distribution to find the k-nearest neighbours with a high probability. Another efficiency improvement comes from the decision when to use arithmetic circuits and when to use polynomials for non-linear operations. Since multiplications in BGV are expensive, the decision on whether to use a polynomial or an arithmetic circuit depends on the number of multiplication gates (size(C)) and the longest path of multiplications (depth(C)) in the circuit. Paterson et al. [55] showed that it is possible to express any polynomial $\mathbb{P}_p(x) : \mathbb{Z}_p \to \mathbb{Z}_p$ with an arithmetic circuit C, such that size(C) = $\mathcal{O}(\sqrt{p})$ and depth(C) = $\mathcal{O}(\log p)$. Thus, depending on the complexity of the polynomial to compute, Shaul et al. are able to decide whether to use polynomials or an arithmetic circuit.

Jäschke and Armknecht [38] focused on an unsupervised version of a k-NN problem, k-Means clustering in the standalone server scenario. The authors use a fully homomorphic encryption scheme to perform linear and non-linear operations on a single cloud server. In order to perform non-linear operations using fully homomorphic encryption, an arithmetic circuit is used. In standard arithmetic circuits, integers are binary encoded. Jäschke and Armknecht experimented using different encodings for arithmetic circuits, and compared the number of additions and multiplications needed. According to [37], using a base $p = 2$ has the best performance. Therefore, the authors use TFHE [76], a fully homomorphic encryption scheme, which uses a binary encoding.

Jäschke and Armknecht also consider using a different distance metric than Euclidean distance. Performing operations on fully homomorphic encrypted data has a significant overhead, therefore one should consider reducing the number of bits used in the computation. The disadvantage of using Euclidean distance is that the distance is squared, which requires more bits for distance computation. Therefore, the authors use the Manhattan distance. Instead of squaring the distance across each dimension, one takes the absolute value. Since a different distance metric is used, misclassification can occur. Jäschke and Armknecht tested the accuracy loss between the Manhattan distance and

Euclidean distance and concluded that less than 2% is misclassified and sometimes performs better. Besides that, [1] claim that Manhattan distance performs better on highly dimensional data.

3.4. Summary

All the works discussed aim to provide privacy without reducing utility, we briefly summarize the techniques. An overview of all the techniques are given in Table 3.4.

Anonymization techniques, such as k-anonymity, l-diversity and t-closeness, make changes to the data, such that it is not possible to identify groups or individuals and link it to sensitive data. The advantage of using anonymized data is being able to perform algorithms in plaintext. However, anonymizing datasets is not a trivial task and cannot guarantee any information leakage, as well as reducing accuracy, since the anonymized data differs from the original data set.

Compression techniques are able to provide security in a standalone server scenario, while achieving sublinear time complexity. However, it has been shown that Bloom Filters are not secure, although Locality Sensitive Hashing is. The disadvantage of using compression techniques is the fact that it is probabilistic, thus it is possible for two unrelated records to be returned.

Cryptographic techniques are able to provide security by encrypting the data, while still being able to perform operations on the ciphertext. The main disadvantage of using cryptographic techniques is the computational or communicational overhead. As discussed before, some works use multiple cloud servers or the client to help with computing intermediate results to increase efficiency.

All in all, to the best of our knowledge, there does not exist a privacy preserving k-NNS protocol which is suitable for outsourced environments which is deterministic in the standalone server scenario and with minimal interaction.

Table 3.4: Overview of related work

Work	Secure	Deterministic	Standalone Server	k-NNS	Approach
Anonymization [68][51][48]	✖	✖	✓	✓	Anonymization
Vatsalan and Christen [72]	✖	✖	✓	✓	Bloom filter
Huang et al. [31]	✓	✖	✓	✓	LSH
Eldmehdwi et al [23]	✓	✓	✖	✓	Paillier
Kim et al. [43]	✓	✓	✖	✓	Paillier
Kesarwani et al. [43]	✓	✓	✖	✓	BGV
Zheng et al. [77]	?	✖	✖	✓	MASPE
Chen et al. [14]	✓	✖	✖	✓	Garbled Circuits
Shaul et al. [63]	✓	✖	✓	✖	BGV
Jäschke and Armknecht [38]	✓	✓/✖	✓	✖	TFHE

4

Privacy Preserving k-NNS

In this chapter, we introduce our Privacy Preserving k-NNS protocol. First we introduce the goals of our protocol, secondly we describe which privacy preserving technique is used. Next, the setup of our protocol is discussed, as well as how to perform distance calculation and top-k selection. Any syntax that is used during this chapter is described in Table 4.1.

Table 4.1: Syntax used in the chapter

Symbol	Definition
$[[\cdot]]$	Homomorphically encrypted value
$min([[\cdot]], [[\cdot]])$	Min function on two encrypted values
$max([[\cdot]], [[\cdot]])$	Max function on two encrypted values
$gt([[\cdot]], [[\cdot]])$	Greater than function on two encrypted values and result $\in \{0, 1\}$
$eq([[\cdot]], [[\cdot]])$	Equal function on two encrypted values and result $\in \{0, 1\}$
$CMUX([[c]], [[l]], [[r]])$	A multiplexer circuit for encrypted data, which returns either l or r, depending on c

4.1. Design Goals

To design a Privacy Preserving k-NNS, it is necessary to establish requirements the protocol should conform to. We briefly describe the requirements:

- **Query Privacy**: The protocol should guarantee query privacy of the client. Query privacy requires that only the client is able to read the contents of the query. Therefore, the server and any third-parties cannot read the contents of the query.
- **Result Privacy**: The result of the protocol, from the server to the client, should be protected such that only the client is able to read the contents. The server or any third-parties should not be able to read the contents.
- **Data Privacy**: The data stored on the server, given out by the data owner, should be kept private for the server, as well as any third-parties. Besides that, the client should only be able to read the contents of the result and not any other data kept on the server.
- **Parallelism**: The protocol should be highly parallelizable. Parallelism is desired, since cloud servers typically provide instances with high number of cores.

- **Standalone Server**: The protocol should work in the standalone server scenario, which requires minimal interaction between the client and a single server. The server should perform all the computations and is not allowed to outsource the computation to other servers.
- **Deterministic**: The protocol should be deterministic, meaning it should provide consistent results, comparable to plaintext implementations.

For our protocol, it is also necessary to define the adversarial behaviour and capabilities. As described in Section 2.3.1, our protocol will be secure against bounded adversaries, since the security of TFHE also depends on bounded adversaries. Besides that, our protocol will be secure against semi-honest adversarials, as described in Section 2.3.2, which is inline with other works on privacy preserving k-NNS.

4.2. Setup

Before the client can send queries to the server, some set up is required by the client. The client is responsible for choosing the parameters of the encryption scheme and generating and distributing the keys.

As discussed earlier, TFHE performs arithmetic circuits on a bitwise level. In order to represent large integers, TFHE concatenates multiple smaller ciphertexts using either radix-based integers or the Chinese Remainder Theorem (CRT). In our work, we only consider the radix-based integers. Although CRT is faster in some instances, due to faster bootstrapping and not needing to propagate carries in circuits, it does not support comparisons, min or max functions. Comparisons, min and max functions are required to perform our protocol.

Instead, a radix representation is used, which concatenates smaller blocks together to eventually represent a larger integer. The number of blocks needed to represent an integer depends on the size of the smaller blocks, which are called shortints. Shortints consist of a message space and a buffer space, where both the message and the buffer space support a size from 1 to 4, which is called the basis $\mathcal{B}$. The message space holds the actual message, where the buffer space is needed to propagate results to the next shortint. To represent a large integer, the integer can be written as $m = m_0 + m_1 * \mathcal{B} + m_2 * \mathcal{B}^2 + ...$, where each m_i is encrypted individually and $m_i < \mathcal{B}$. The size of the basis $\mathcal{B}$ has impact on the efficiency of the operations, therefore it is important to choose the right basis. Although the message basis and the buffer basis can differ, we will keep the buffer and the message the same size. If the buffer space is smaller than the message space, it can lead to incorrect results due to over-and underflows.

Jäschke and Armknecht argue that using a basis of 2 leads to the best overall performance [37]. Besides that, the authors of the implementation of TFHE published benchmarks of different operations using different parameters, see Table 4.2. However, Table 4.2 only considers operations on a shortint using different parameters. Therefore, Table 4.3 shows the performance of different parameters using 16-bit integers. In the case of using a radix base of 3, we used an 18 bit integer, since 16 is not divisible by 3. As seen in Table 4.3, only a basis of 1 is faster than a basis of 2, just for addition. Not only is basis 2 faster in multiplications, but using a basis of one, the max function is not supported due to the need to perform programmable bootstrapping, which requires at least a message + buffer space of 4. All in all, the parameter set where the message and buffer space consist of 2 bits performs the best. When using shortints with $\mathcal{B} = 2$, the number of blocks needed to represent an integer is $n_{blocks} = \frac{n_{bits}}{2}$, where n_{bits} is the size of the plaintext message space.

Table 4.2: Benchmark performances of different message and buffer sizes for TFHE [76]

Parameter set	add	mul_lsb	programmable_bootstrap
PARAM_MESSAGE_1_BUFFER_1	7.90 ms	8.00 ms	8.10 ms
PARAM_MESSAGE_2_BUFFER_2	18.4 ms	18.1 ms	18.4 ms
PARAM_MESSAGE_3_BUFFER_3	131.5 ms	129.5 ms	129.4 ms
PARAM_MESSAGE_4_BUFFER_4	852.5 ms	839.7 ms	828.1 ms

The client is responsible for generating the keys and providing the server with the keys required to perform the protocol. The client should generate three keys: Secret Key, Public Key and Cloud Key. The secret key is used to encrypt and decrypt data and should always be kept private from other parties.

Table 4.3: Total running time summed for 1000 operations of different message and buffer sizes using 16-bit integers. (For PARAM_MESSAGE_3_BUFFER_3 an 18 bit integer was used)

Parameter set	add	mul	max
PARAM_MESSAGE_1_BUFFER_1	11.68 s	29.98s	-
PARAM_MESSAGE_2_BUFFER_2	13.09 s	21.80 s	25.21 s
PARAM_MESSAGE_3_BUFFER_3	60.00 s	106.40 s	115.82 s
PARAM_MESSAGE_4_BUFFER_4	181.68 s	>500 s	368.27 s

Leaking the secret key allows any arbitrary party to decrypt immediate steps in the protocol. The public key can be distributed to the server, which allows the server to only encrypt messages. The cloud key is used to perform homomorphic operations on the encrypted data and to perform bootstrapping. After the client has generated the keys, the client should encrypt all the data using the secret key and send the encrypted data to the server, together with the public and cloud key.

The security of the scheme depends on parameters chosen, such as noise level and key size. We assume parameters are chosen such that it is able to provide 128 bits of security [76], calculated by a lattice estimator.

4.3. Distance Calculation

The first step in performing k-NNS using Fully Homomorphic Encryption (FHE) is calculating the distances between the query and the records in the database. Distances can be defined by using a distance metric, which formulates a function to define a distance between two points. We will briefly describe two commonly used distance metrics, then we perform experiments to find the accuracy loss between the two. Finally, we discuss the actual implementation using fully homomorphic encryption of the distance metric.

4.3.1. Distance Metric

The most common metric is the Euclidean distance, as defined in Equation 4.1. In short, Euclidean sums the squared distances of all the attributes and then takes the square root. The advantage of using Euclidean distance for distance metric, is that outliers are move heavily punished due to the squared exponential. The disadvantage of using Euclidean distance is also the squared factor, which causes the distance to grow exponentially. If the distance space grows faster, this means that more bits are needed to represent the distance without an overflow. Besides that, performing a square root on encrypted data is computationally expensive and not accurate. Therefore, literature uses Euclidean distance without the square root, as defined in Equation 4.2

$$d(x,q) = \sqrt{\sum_{j=0}^{j=d} (x_j - q_j)^2} \tag{4.1}$$

$$d(x,q) = \sum_{j=0}^{j=d} (x_j - q_j)^2 \tag{4.2}$$

Another example of a distance metric, is the Manhattan distance as defined in Equation 4.3. Manhattan distance differs from Euclidean distance by taking the sum of the absolute differences instead of squaring it. Using Manhattan distance, outliers are not punished more heavily, but instead have the same weighting as points close the query. One of the reasons of using Manhattan distance is reducing the distance space, as done in [38]. Since Manhattan distance uses an absolute function instead of squaring, the distance space grows linearly. Therefore, when using the Manhattan distance metric, it is possible to use less bits in the protocol than the Euclidean distance metric.

$$d(x,q) = \sum_{j=0}^{j=d} |x_j - q_j| \tag{4.3}$$

The main cost of homomorphic operations using TFHE is the bootstrapping operation between shortint block, which is necessary to propagate the carry between blocks. Therefore, performing operations on a smaller number of bits is preferred, since this results in less bootstrapping operations. That is why our protocol will implement the Manhattan distance.

4.3.2. Implementation

Now that we have defined the distance metric, the Manhattan distance, it needs to be implemented using fully homomorphic encryption. We will discuss challenges and solutions for implementing the Manhattan distance.

As mentioned in Section 2.4, we will be making use of TFHE [76] as homomorphic encryption scheme. TFHE only supports unsigned integers, which is a challenge since both Euclidean and Manhattan distance requires negative numbers.

The first solution requires using an encoding such that it is possible to represent signed integers. An earlier work by Jäschke and Armknecht [38] used two's complement, together with an arithmetic circuit to implement the absolute function. The algorithm is shown in Algorithm 1. First, the most significant bit c and the negation of a are calculated. Then, using a controlled multiplexer, depending on the most significant bit c, either x' or x is returned. This circuit works, since if the most significant bit is set, x is already negative. Therefore, returning the negation of x, which was already calculated by x', the positive representation of x is returned. If c is not set, then x is already positive and return x.

Algorithm 1 Manhattan distance using absolute arithmetic circuit proposed by Jäschke and Armknecht [38].

Input:
$[[a]] = [[a_n]] \dots [[a_1]][[a_0]]$: binary encoded encrypted integer
$[[b]] = [[b_n]] \dots [[b_1]][[b_0]]$: binary encoded encrypted integer
Output: $[[|a - b|]]$
$[[x]] \leftarrow [[a]] - [[b]]$
// Set the control bit as the most significant bit of x
$[[c]] \leftarrow [[x_n]]$
// Calculate the negation of x
$[[x']] \leftarrow -[[x]]$
// If $c = 1$, then x is negative, thus return x'. Else return x
$[[d]] \leftarrow CMUX([[c]], [[x']], [[x]])$
return $[[d]]$

Although the absolute algorithm proposed by Jäschke and Armknecht works, it has disadvantages. Using two's complement reduces the message space by one bit, since it is required to represent negative numbers. This is unfavourable, because each bit that is required increases the runtime. Besides that, calculating the circuit as described in Algorithm 1 is computationally expensive. The circuit is computationally expensive because of the CMUX, which is a gate that uses a control bit c, to select either x or y. CMUX gates are useful, because it is not possible to perform if else branches on encrypted data. Since the data is encrypted, it is impossible to branch and therefore both branches have to be executed. However, it is still possible to implement if else branches by using a CMUX gate, by multiplying by 1 if a statement is true, and 0 when a statement is false and summing the result. A CMUX can be written in multiple ways, as shown in Equation 4.4 and 4.5. Using Equation 4.5 is preferred over Equation 4.4, since this saves doing an extra multiplication. As mentioned before, multiplications are computationally expensive and therefore should be avoided as much as possible.

$$CMUX(c, x, y) = (1 - c) * x + c * y \tag{4.4}$$

$$CMUX(c, x, y) = c * (x - y) + y \tag{4.5}$$

We therefore propose an arithmetic circuit using TFHE, which is more efficient in calculating the Manhattan distance. According to the benchmarks published by Zama [76], which are shown in Table 4.4,

multiplications become increasingly more expensive as the bit size increases. While, performing a minimum function only becomes slightly more expensive when the bit size increases. Therefore, when using a plaintext size of 16 bits, it is faster to perform two min or max functions than one multiplication. Since, Algorithm 1 uses at least one multiplication in the CMUX, we propose to calculate the Manhattan distance using a min or a max. Our implementation is shown in Algorithm 3. By first calculating the maximum of two integers and the minimum of two integers, it is possible to guarantee that $left - right \geq 0$, since $left \geq right$. Not only is our solution faster, we do not require an encoding which requires the use of an extra bit.

Table 4.4: Performance benchmarks for the add, mul and min operations using TFHE [76]. Each radix-based integer consists of two message bits and two buffer bits and using the default flag.

Plaintext Size	add	mul	min
8 bits	129.0 ms	227.2 ms	186.8 ms
16 bits	256.3 ms	756.0 ms	233.1 ms
32 bits	469.4 ms	2.10 s	282.9 ms
64 bits	959.9 ms	5.53 s	336.5 ms

Algorithm 2 Our Manhattan distance arithmetic circuit.

Input:
$[[a]] = [[a_n]] \ldots [[a_1]][[a_0]]$: binary encoded encrypted integer
$[[b]] = [[b_n]] \ldots [[b_1]][[b_0]]$: binary encoded encrypted integer
Output: $[[\ |a - b|\ ]]$
// Find the maximum of a and b
$[[left]] \leftarrow max([[a]], [[b]])$
// Find the minimum of a and b
$[[right]] \leftarrow min([[a]], [[b]])$
// d is always ≥0, since $left \geq right$
$[[d]] \leftarrow [[left]] - [[right]]$
return $[[d]]$

4.4. Top-k Selection

The next step in performing Privacy Preserving k-NNS, is finding the minimum k-distances. This section discusses different approaches to find the minimum of a set of distances.

4.4.1. Sorting

The trivial implementation of finding the k minimum distances is to first sort the distances and return the k first entries. However, secure sorting algorithms on encrypted data are inefficient. Commonly used sorting algorithms such as quick sort or merge sort cannot be performed securely without significant overhead, due to not being able to partition the data. The data cannot be partitioned, since two ciphertexts are indistinguishable from each other and can therefore not decide on how to partition the data. Therefore, quick sort and merge sort have the same complexity as bubble and insertion sort, $\mathcal{O}(n^2)$. Bitonic and odd-even merge sort are sorting algorithms which can be performed parallel and be performed on encrypted data, with a complexity of $\mathcal{O}(n\log_2(n))$. Although this is more efficient than other sorting algorithms, the algorithm performs the work to sort all the distances, while only k are needed. Therefore, we believe we can reduce the complexity in the case of Privacy Preserving k-NNS. An overview of the complexities of the sorting algorithms are shown in Table 4.5.

4.4.2. Plaintext Solutions

Finding the smallest or largest k items in a list is a well solved problem in the plaintext setting, however we will show these do not apply when working on encrypted data.

Hadian and Sobel already showed in 1969 that the upper bound for the amount of comparisons for finding the k largest items is bounded by Equation 4.6, where n is the total number of items. Hadian

Algorithm 3 Our distance computation circuit

Input:
$Q = Q \in \mathbb{Z}^d$: Encrypted query
$D = D \in \mathbb{Z}^{n \times d}$: Encrypted database
Output: $distances$: Encrypted distances
$distances \leftarrow Empty$
// Do for each row in the database
for i ←0, …n **do**
// initialize distance to 0 for each row
$[[localDist]] \leftarrow [[0]]$
$x \leftarrow D[i]$
// Do for each attribute
for j ←0, …, d **do**
$[[a]] \leftarrow x[j]$
$[[b]] \leftarrow Q[j]$

// Calculate Manhattan distance between a and b
$[[left]] \leftarrow max([[a]], [[b]])$
$[[right]] \leftarrow min([[a]], [[b]])$
$[[dist]] \leftarrow [[left]] - [[right]]$

// Sum the distance
$[[localDist]] \leftarrow [[localDist]] + [[dist]]$
end for
$distances.push([[localDist]])$
end for
return $distances$

Table 4.5: Complexity of sorting algorithms on encrypted data, expressed as the number of comparisons needed. (Retrieved from [25])

Sorting Algorithm	Complexity
Bubble Sort	$\mathcal{O}(n^2)$
Insertion Sort	$\mathcal{O}(n^2)$
Merge Sort	$\mathcal{O}(n^2)$
Insertion Sort	$\mathcal{O}(n^2)$
Bitonic Sort	$\mathcal{O}(n(\log_2(n))^2)$
Odd-Even Sort	$\mathcal{O}(n(\log_2(n))^2)$

and Sobel were able to prove this using replacement selection in a binary tree, and will briefly discuss this to find the k minimum values. First, create a binary tree with $n-k$ items, and keep those k items in reserve. After finding the minimum in the tree, and replace the minimum of that we found in the leave of the tree, with one of the items we kept in reserve and find the minimum again. However, this time we do not have to compare everything in the tree, since we already computed previous paths. Only compute the path from the leave to the root that we changed, resulting in needing to perform fewer comparisons. Although, replacement selection provides us with a good upper bound for the number of comparisons, it cannot be efficiently performed on encrypted data, because replacement selection requires partitioning. To perform replacement selection on encrypted data would require to compute all the paths in the tree again, losing the efficiency of the algorithm.

$$V_k \leq n - k + (k-1)\lceil log_2(n+2-k) \rceil \tag{4.6}$$

Another solution involves using a min or max heap data structure to hold the k highest or lowest items in the list. A heap is a tree like structure which allows to efficiently search the tree for smaller or larger items. When iterating over distances, compare if the distance is smaller than the largest item in the heap and replace it. In plaintext this can be implemented in $\mathcal{O}(n\log_2(k))$. Again, using

such an implementation introduces challenges when converting it to work on encrypted data. Firstly, searching the heap introduces additional complexity since it requires partitioning, which is not possible on encrypted data. Secondly, using a min heap structure is not parrallelizable, since it requires editing a single data structure which would require to work with a mutex, thus slowing down computation.

4.4.3. Our Solution

Our solution is a variant of the replacement selection algorithm as described in Section 4.4.2. The first step in our algorithm is to create a binary tree, to compute all the distances to find the minimum distance. Using the $min(x, y)$ function, the smallest value is propagated to the root of the tree. For the trivial case where $k = 1$, the root of the tree contains the smallest value of all the values in the list, and the ID is returned. The advantage of using a binary tree, is the possibility of parallelizing the computation as much as possible.

For the case when $k > 1$, it is necessary to perform another step to find all the other k's. After a minimum has been found, it needs to be removed from the leave of the tree and replaced by an encryption of MAX_VALUE. One way to replace the minimum is to iterate over the list of distances again, and check if the ID of the minimum is equal to the one in the list, which is the most accurate solution. The entire algorithm is shown in Algorithm 4.

Note that performing an equality function on two encrypted values, returns an encrypted bit over the entire plaintext domain. Therefore, when performing operations which multiply values from an equality, only the least significant shortint block of the output integer of the equality function has to be used. Using only the least significant shortint block during multiplications can be used in for example a CMUX circuit to speed up computation.

4.4.4. Optimization

Algorithm 4 shows the implementation on how to retrieve the ID of the k closest records, but we will show it is possible to make some optimizations in some special scenarios where only some specific attributes need to be returned. The disadvantage of using the ID as a replacement attribute, is that the size of the data set size is limited by the size of the plaintext domain, which is defined as: $\mathcal{B}^2 *$num_of_blocks $> n$, where $\mathcal{B}$ is the message size for a shortint and n the data set size. The database limit can be increased by incrementing either the message size of the shortint or the number of blocks in an integer, but increasing the number of bits or blocks impacts the efficiency of the protocol significantly. Therefore, if it is possible to remove the ID attribute from the protocol, and only return the specific attributes needed, without increasing the bit size of the shortints or number of blocks.

Using the distance of the minimum found in a round instead of the ID, it is possible to update the list of distances, which is shown in Algorithm 5. The advantage of using the distance instead of the ID, is now we can support large databases without compensating for a large bit size for the plaintext. The disadvantage of using such a method, is that the distances need to be unique to achieve a 100% accuracy. If the distances are not unique, it can occur that two records with the same respective distance to a query get both updated instead of one, resulting in a different result than a plaintext implementation.

Updating only one record is possible with an extra variable, as shown in Algorithm 6, however this introduces extra overhead due to the need for extra circuits. The downside of using additional circuits, is that gates are expensive and also make the protocol less parrallelizable due to the need for a mutex. Because mutex is a data structure such that only one thread can have access to a resource at a time, this results in extra overhead for threads that cannot access the data structure. However, using an extra bit in combination with a mutex makes it possible to achieve the same accuracy as the algorithm comparing ID's.

An encrypted bit is used, which is initialized as 0, and will be set to 1 when the minimum distance has been found in the list of distances, which is done by using an OR gate on the bit which check for equality of the minimum distance. Then the CMUX is only applied when the equality check bit is set and the encrypted bit is not set.

Algorithm 4 Top-k Selection using ID as replacement

Input:
$distances = [([[ID, dist]]_1), \ldots, ([[ID, dist]]_n)]$: Homomorpically encrypted tuples of the record ID and distance
Ck : Cloudkey to perform homomorphic operations
Pk : Publickey to encrypt plaintext to ciphertext
k : The number of nearest neighbours to find
Output: $[([[ID, dist]]_1), \ldots, [[ID, dist]]_k]$ The top-k closest records to a specific query

$topK \leftarrow Empty$
// Start a new round to find a new minimum value
for $i \leftarrow 1, \ldots, k$ **do**
  $dist \leftarrow distances$
  // Find the minimum value in the list of distances
  while $size(dist)! = 1$ **do**
    $[[left]] \leftarrow [[dist]].pop()$
    $[[right]] \leftarrow [[dist]].pop()$
    $[[low]] \leftarrow min([[left]], [[right]])$
  end while
  // The found minimum distance
  $[[min]] \leftarrow [[dist]].pop()$
  $topK.push([[min]])$
  // If in the last round, skip the last phase
  if $i = k$ **then**
    break
  end if
  // Update list of distances for a new round
  for $j \leftarrow 1, \ldots, n$ **do**
    $([[ID]], [[dist]])) \leftarrow distances[j]$
    $[[c]] \leftarrow Ck.eq([[ID]], [[min.dist]])$
    $[[maxDistance]] \leftarrow Pk.encrypt(MAX\_VALUE)$
    // Replace the distance of this record with MAX_VALUE if IDs match
    $[[newDist]] \leftarrow CMUX([[c]], [[dist]], [[maxDistance]])$
    $distances[j] \leftarrow ([[ID]], [[newDist]])$
  end for
end for
return $topK$

Algorithm 5 Top-k Selection using the distance as replacement.

Input:
$distances = [([[attr_i, dist]]_1), \dots, ([[attr_i, dist]]_n)]$: Homomorpically encrypted tuples of the record ID and distance
Ck : Cloudkey to perform homomorphic operations
Pk : Publickey to encrypt plaintext to ciphertext
k : The number of nearest neighbours to find
Output: $[([[attr_i, dist]]_1), \dots, [[attr_i, dist]]_k]$ The top-k closest records to a specific query

$topK \leftarrow Empty$
// Start a new round to find a new minimum value
for $i \leftarrow 1, \dots, k$ **do**
  $dist \leftarrow distances$
  // Find the minimum value in the list of distances
  while $size(dist)! = 1$ **do**
    $[[left]] \leftarrow [[dist]].pop()$
    $[[right]] \leftarrow [[dist]].pop()$
    $[[low]] \leftarrow Ck.min([[left]], [[right]])$
  end while
  // The found minimum distance
  $[[min]] \leftarrow [[dist]].pop()$
  $top - k.push([[min]])$
  // If in the last round, skip the last phase
  if $i = k$ **then**
    break
  end if
  // Update list of distances for a new round
  for $j \leftarrow 1, \dots, n$ **do**
    $([[attr_i]], [[dist]])) \leftarrow distances[j]$
    $[[c]] \leftarrow Ck.eq([[dist]], [[min.dist]])$
    $[[maxDistance]] \leftarrow Pk.encrypt(MAX\_VALUE)$
    // Replace the distance of this record with MAX_VALUE if distances match
    $[[newDist]] \leftarrow CMUX([[c]], [[dist]], [[maxDistance]])$
    $distances[j] \leftarrow ([[attr_i]], [[newDist]])$
  end for
end for
return $topK$

Algorithm 6 Top-k Selection using distance as replacement together with a mutex

Input:
$distances$: Homomorpically encrypted tuples of the record ID and distance
Ck : Cloudkey to perform homomorphic operations
Pk : Publickey to encrypt plaintext to ciphertext
k : The number of nearest neighbours to find
Output: $[([[attr_i, dist]]_1), \dots, [[attr_i, dist]]_k]$ The top-k closest records to a specific query

$topK \leftarrow Empty$
// Start a new round to find a new minimum value
for $i \leftarrow 1, \dots, k$ **do**
  $dist \leftarrow distances$
  // Find the minimum value in the list of distances
  while $size(dist)! = 1$ **do**
    $[[left]] \leftarrow [[dist]].pop()$
    $[[right]] \leftarrow [[dist]].pop()$
    $[[low]] \leftarrow Ck.min([[left]], [[right]])$
  end while
  // The found minimum distance
  $[[min]] \leftarrow [[dist]].pop()$
  $top - k.push([[min]])$
  // If in the last round, skip the last phase
  if $i = k$ **then**
    break
  end if
  // Encrypt a 0 bit to keep count if a replacement has been found
  $[[m]] \leftarrow Pk.encrypt(0)$
  // Update list of distances for a new round
  for $j \leftarrow 1, \dots, n$ **do**
    $([[attr_i]], [[dist]])) \leftarrow distances[j]$
    $[[c]] \leftarrow Ck.eq([[dist]], [[min.dist]])$
    // Only perform replacement when distances are equal and *m* == 0
    $select \leftarrow Ck.gt([[c]], [[m]])$
    // Update tracking bit if a replacement has been found or has already been found
    $[[m]] \leftarrow Ck.or([[ml]], [[c]])$
    $[[maxDistance]] \leftarrow Pk.encrypt(MAX\_VALUE)$
    // Replace the distance of this record with MAX_VALUE if *select* == 1
    $[[newDist]] \leftarrow CMUX([[select]], [[dist]], [[maxDistance]])$
    $distances[j] \leftarrow ([[attr_i]], [[newDist]])$
  end for
end for
return $topK$

5

Analysis

This chapter further analyses our protocol, using theoretical and practical analyses. First, in the theoretical analysis, we discuss the security of our protocol and the complexity. Finally, in the practical analysis, our protocol is run on a cloud server to test the scalability of our protocol with respect to the number of cores.

5.1. Theoretical Analysis

5.1.1. Security Analysis

To prove that our Privacy Preserving k-NNS protocol is secure, it is important to establish that the underlying cryptographic primitive, TFHE, is secure under a certain set of parameters, by using Lemma 1 and 2. After we have established that the underlying cryptographic primitive is secure, we show that our protocol is secure under a semi-honest client and server. To prove Theorem 1, we show that our protocol holds under a simulation proof.

The hardness of the TFHE scheme, relies on the hardness of a torus based problems of Learning With Errors (LWE) [11] and Generalized Learning With Error (GLWE) [15]. The hardness of both LWE and GLWE are described in Lemma 1 and 2 respectively, for some security parameter λ, where $n := n(\lambda)$. $\mathcal{X} := \mathcal{X}(\lambda)$, $N := N(\lambda)$. $\mathcal{X} := \mathcal{X}(\lambda)$ and $k = k(\lambda)$ [17]. Since ciphertexts are indistinguishable from random noise under a sufficiently large λ, TFHE is IND-CPA secure with 128 bits of security [75].

Lemma 1. *(LWE problem over the torus). Let $n \in \mathbb{N}$ and let $s = (s_1, \dots, s_n) \xleftarrow{\$} \mathcal{B}^n$. Let also $\mathcal{X}$ be an error distribution over $\mathcal{R}$. It is computationally hard to distinguish the following distributions [17]:*

$\mathcal{D}_0 = \{(a, r) | a \xleftarrow{\$} \mathbb{T}^n, r \xleftarrow{\$} \mathbb{T}\}$
and
$\mathcal{D}_1 = \{(a, r) | a = (a_1, \dots, a_n) \xleftarrow{\$} \mathbb{T}^n, r = \sum_{j=1}^{n} s_j \cdot a_j + e, e \leftarrow \mathcal{X}\}$.

Lemma 2. *(GLWE problem over the torus). Let $N, k \in \mathbb{N}$ with N a power of 2 and let and let $s = (s_1, \dots, s_k) \xleftarrow{\$} \mathcal{B}_N[X]^k$. Let also $\mathcal{X}$ be an error distribution over $\mathcal{R}_n[X]$. It is computationally hard to distinguish the following distributions [17]:*

$\mathcal{D}_0 = \{(a, r) | a \xleftarrow{\$} \mathbb{T}_N[X]^k, r \xleftarrow{\$} \mathbb{T}_N[X]\}$
and
$\mathcal{D}_1 = \{(a, r) | a = (a_1, \dots, a_k) \xleftarrow{\$} \mathbb{T}_N[X]^k, r = \sum_{j=1}^{k} s_j \cdot a_j + e, e \leftarrow \mathcal{X}\}$.

We will now prove that our privacy preserving k-NNS protocol is secure under the bounded honest-but-curious adversarial model, using a simulation proof. The proof holds for all three versions of our k-NNS protocol, and proves Theorem 1. A simulation proves that and adversary can learn approximately as much information from a ciphertext in the *Real* world, as if it had received nothing or random noise, *Ideal* world.

To set up the simulation proof, we follow the construction from [63] and use Definitions 3 and 4 from [50] to prove the security of our protocol under honest-but-curious adversaries. Let π be a protocol

which executes f and $\text{View}_{\pi,i}(x, y, \lambda)$ the view of a party i, which executes π on (x, y), where λ is the security parameter. The view of a part i, includes its inputs, randomness and all message received. In our case, our protocol consists of two parties, the client and the server. The parties have the following shared parameters:

- $\mathcal{E}$: IND-CPA secure encryption scheme,
- λ: Security parameter,
- d: Number of attributes of a record in the database,
- b: Plaintext domain size,
- Pk: Public key,
- k: Number of items to return,
- Ck: Cloud server key.

Besides that, the client has private inputs:

- Q : Query $Q \in \mathbb{Z}_{2^b}^d$,
- Sk: Secret key,

and outputs k ID's of records in the database: $topK \in \mathbb{Z}_{2^b}^k$. Finally, the server has the following additional input:

- D: Database $D \in \mathbb{Z}_{2^b}^{n \times d}$,
- n: Size of the database,

and has no output.

Definition 3. *Let f be a functionality such that $f : \{0,1\}^* \times \{0,1\}^* \to \{0,1\}^* \times \{0,1\}^*$, where $f = (f_1, f_2)$. That is, for every pair of inputs $x, y \in \{0,1\}^n$, the output pair is a random variable $(f_1(x, y), f_2(x, y))$ ranging over pairs of strings. The first party (with input x) wishes to obtain $f_1(x, y)$, and the second party (with input y) wishes to obtain$f_2(x, y)$*

Definition 4. *Let $f = (f_1, f_2)$ be a functionality. We say that π securely computes f in the presence of honest-but-curious adversary if there exist probabilistic polynomial-time algorithms S_1 and S_2 such that:*
$\{(S_1(1^n, x, f_1(x, y)), f(x, y))\}_{x,y,n} \overset{c}{\equiv} \{(\textit{view}_1^{\pi}(x, y, n), \textit{output}^{\pi}(x, y, n))\}_{x,y,n}$, *and*
$\{(S_2(1^n, x, f_2(x, y)), f(x, y))\}_{x,y,n} \overset{c}{\equiv} \{(\textit{view}_2^{\pi}(x, y, n), \textit{output}^{\pi}(x, y, n))\}_{x,y,n}$,
where $x, y \in \{0,1\}^*$ such that $|x| = |y|$, and $n \in \mathbb{N}$.

Theorem 1. *Assuming the hardness of torus based problems of LWE and GLWE, our k-NNS protocol is secure, under a semi-honest client and server.*

Proof. We construct a simulator $\mathcal{S}$, when given the server's inputs and outputs $(1^\lambda, \mathcal{E}, b, d, k, n, S, topK)$, is computationally indistinguishable from a real execution of the protocol. The simulator performs the following operations:

1. Generate a random query $Q' \in \mathbb{Z}_{2^b}^d$, and encrypt it with the public key.
2. Execute the k-NNS protocol using Q'.
3. The simulator outputs the result of the protocol, topK' $\in \mathbb{Z}_n^{k \times n}$.

The output of the simulator in the *Ideal* scenario is $S(\dots) = ([[Q']], [[topK']])$, which we will now prove is indistinguishable from the view of the server in the *Real* scenario. The view of the server in the *Real* scenario is $view(\mathcal{A}) = ([[Q]], [[topK]])$. Both the view of the simulator and the real scenario are indistinguishable, since both the view of the simulator and server are generated by a IND-CPA secure scheme $\mathcal{E}$. Therefore, we can conclude according to Definition 4 that $S(\dots) \overset{c}{\equiv} view(\mathcal{A})$, thus our protocol is secure against honest-but-curious adversaries.

□

5.1.2. Complexity Analyses

To compare our protocol to other protocols, it is necessary to define the complexity of our protocol, which we will define by giving the runtime complexity.

First, the runtime complexity is discussed expressed in the number of entries in the database, n, the number of attributes each entry has, d, and k. As described in Chapter 4, the protocol consists of three main steps: distance computation, top-k selection, distance replacement. An overview of all the runtime complexities are given in Table 5.1.

The distance computation is equal for all three version of the protocol, as described in Chapter 4. To compute the distance, it requires calculating the Manhattan distance for each record in the database. Computing the Manhattan distance for one record, involves summing the absolute difference between two values, giving a runtime complexity of $\mathcal{O}(d)$. Thus, calculating the Manhattan distance n times, gives a runtime complexity of $\mathcal{O}(n * d)$.

Next, the top-k selection runtime is also equal for all three version of the protocol, since the difference between the protocols are a tradeoff between accuracy, utility and speed. To find the minimum value of a list of values, we create a binary tree and calculate all possible branches of this tree. Requiring us to calculate $(\frac{n}{2} + \frac{n}{4} + \frac{n}{8} + \frac{n}{16} + ...)$, $min()$ functions, giving us a runtime complexity of $\mathcal{O}(n)$. However, since the protocol aims to find the k smallest value, it performs the binary tree k times, giving a runtime complexity of $\mathcal{O}(n * k)$.

After the minimum value was found, it is required to update the minimum value in the list of distances, otherwise in the next iteration, the same value will be found. By iterating over the list of distances and performing an equality check in combination with a CMUX gate, it is possible to update the values. Now, since we iterate over the entire list and repeat $k - 1$ times, it results in a runtime complexity of $\mathcal{O}(n * k)$.

Finally, the total runtime complexity of all three protocols can be defined, by adding the runtime complexities together. First, we perform the distance computation and after this, perform the top-k selection with the distance updates, giving a runtime complexity of: $\mathcal{O}((n * d) + (n * k) + (n * k))$. This can be simplified to: $\mathcal{O}((n * d) + 2(n * k)) \rightarrow \mathcal{O}(n * (d + k))$.

Table 5.1: Runtime complexities for the server for a given query for all three protocols.

Protocol	ID	Distance	Mutex
Distance Computation	$\mathcal{O}(n * d)$	$\mathcal{O}(n * d)$	$\mathcal{O}(n * d)$
Top-k Selection	$\mathcal{O}(n * k)$	$\mathcal{O}(n * k)$	$\mathcal{O}(n * k)$
Update Distances	$\mathcal{O}(n * k)$	$\mathcal{O}(n * k)$	$\mathcal{O}(n * k)$
Total Complexity	$\mathcal{O}(n * (d + k)))$	$\mathcal{O}(n * (d + k)))$	$\mathcal{O}(n * (d + k)))$

5.2. Practical Analyses

To test the performance of our protocol, the three protocols will be tested again two commonly used datasets, SIFT1M [39] and Deep1B [5]. First, the setup is discussed for the experiments, followed by the results of the experiments.

5.2.1. Setup

Before the experiments are run, it is important to define the metrics such that we can assess the performance of our protocols. Firstly, it is important to check the accuracy of the protocol against a plaintext implementation. Accuracy will be divided into two sub-categories: ordered accuracy and unordered accuracy. Order accuracy will be defined as the number of IDs correct in the right place, divided by the total number requested, k. Unordered accuracy will be determined by the number of IDs correct in any order, divided by the total number requested, k. Besides that, it is important to quantify the execution time of our protocols, and will therefore time how long it takes to execute a query. The execution time is split up into three phases: distance computation, top-k selection and updating distances.

Secondly, the parameters used impact the performance of our protocol and therefore an overview of all parameters used are shown in Table 5.2. We use $k = 50$, such that we can test the accuracy of Algorithm 5. Unfortunately, because $k = 50$, the experiments are computationally expensive and

therefore can only perform one query per Algorithm on a given data set, since access computational resources are limited. In our experiments, the plaintext domain will consist of 16 bits, using 8 blocks of 2 bit shortints, since reducing the number of bits, also reduces the number of operations and bootstrapping needed. The plaintext size of 16 bits was chosen, such that we can guarantee that the distance between any given query does not overflow, while keeping the plaintext size small enough to prevent expensive computations.

The SIFT1M and Deep1B database consist of a million and a billion entries respectively, which is unfeasible to run in this study. Therefore, a subset of these datasets are used, which consist both of the first 1000 entries, having 128 and 96 attributes respectively. Together with the dataset, example queries are given, for which the first entry is used in the experiments. The values in the SIFT1M dataset are already integers between 0 and 255 and thus requires no normalizing, since we can guarantee that our underlying plaintext does not overflow, since $128 * 255 < 2^{16}$. However, the values in the Deep1B dataset are floating point numbers between -1 and 1, which requires normalizing before use. Our protocol only considers positive integers, thus first the data set is incremented by one, afterwards divided 2, such that the values now lie between 0 and 1. In order to assure the underlying plaintext does not overflow, the scaling factor should be smaller than: $\frac{2^{16}}{96}$. Using a scaling factor of 9 bits and rounding down, $2^9 = 512$, the highest accuracy is achieved, while assuring correctness. Therefore, all values of the Deep1B dataset are integers between 0 and 512.

All experiments were run on Amazon Computing Services (AWS) [2] EC2 instances. Specifically, the c6ax24large instances, having 96 vCPUs and 192 GiB of random access memory.

Table 5.2

Parameter	shortint_size	n_blocks	k	num_of_queries
Value	2M2B	8	50	1

5.2.2. Results

The results for the subsets of the SIFT1M and Deep1B data set in shown in Tables 5.3 and Table 5.4. The results between the SIFT1M and Deep1B data sets are very similar in accuracy and runtime, except for the distance computation, which depends on the number of entries n, and the number of attributes d. The difference between distance computation time is caused by the large number of attributes in the SIFT1M over the Deep1B data set, 128 over 96. Note that the runtime for each protocol for a given data set are the same, since only the top-k selection part of the protocol differentiates. Besides that, the runtimes of the SIFT1M and the Deep1B data sets are similar, which is explained by the fact that the top-k selection part depends on the number of entries n and k, which are $n = 1000$ and $k = 50$ for both data sets. It is also noteworthy to see that the runtime of the top-k selection step increases linearly with k, see Figure 5.2 and Figure 5.1.

The ID protocol performs as expected, achieving 100% accuracy compared to the plaintext implementation also using the Manhatten distance, as well as having a similar or lower runtime compared to the other two protocols. However, as discussed in Section 4, the ID protocol requires the plaintext size to be able to support the number of entries in the database. Increasing the plaintext size, results in extra computation in all steps of the protocol.

Instead of using the ID's to replace the minimum value in the list of distance, it is possible to match on the distances, for which the accuracy and runtime is also shown in Table 5.3. Using the distance as a matching value has a similar runtime as using the ID value, but decreases the accuracy of the protocol. The accuracy decrease depends on the number of entries in the database that have the same distance to the query, as well as the size of k and the number of entries in the database itself, since the possibility that two entries have the same distance increases as the database size increases. The accuracy decrease for the first 1000 entries in the SIFT1M database as k increases, are shown in Figure 5.3, which shows that after $k = 20$, the accuracy starts to decline. The accuracy for the Deep1B data set shows similar results, where the accuracy decreases after $k = 11$ 5.4.

By using the Mutex protocol, it is possible to have the same accuracy as the ID protocol, while limiting the plaintext size, by sacrificing parallelization and extra overhead. The Mutex protocol requires that only one thread is allowed to obtain the lock, resulting in other threads doing nothing. Besides that, the protocol has extra overhead due to the need to hold an extra bit to determine if the distance has already

been updated. However, due to an implementation bug in the TFHE library which caused deadlocks, we were unable to parallelize the step where the distances are updated, resulting in a significantly larger runtime. Therefore, newer versions of the TFHE library can significantly improve the run time of the protocol. In order to save computation resource, the experiment was stopped after 8 hours of running. Since our protocol scales linearly with k, the graph is extrapolated using the average runtime.

Table 5.3: Runtime and accuracy for the first 1000 entries in the SIFT1M dataset and k = 50.

Protocol:	ID	Distance	Mutex
Ordered accuracy	100%	40%	100%
Non-ordered accuracy	100%	92%	100%
Runtime distance computation	124 mins	124 mins	124 mins
Runtime top-k selection	167 mins	165 mins	23.48 hours*
Total runtime	291 mins	279 mins	25.55 hours*

Table 5.4: Runtime and accuracy for the first 1000 entries in the Deep1B dataset and k = 50.

Protocol:	ID	Distance	Mutex
Ordered accuracy	100%	22%	100%
Non-ordered accuracy	100%	86%	100%
Runtime distance computation	92 mins	92 mins	92 mins
Runtime top-k selection	165 mins	166 mins	23.62 hours*
Total runtime	257 mins	258 mins	25.16 hours*

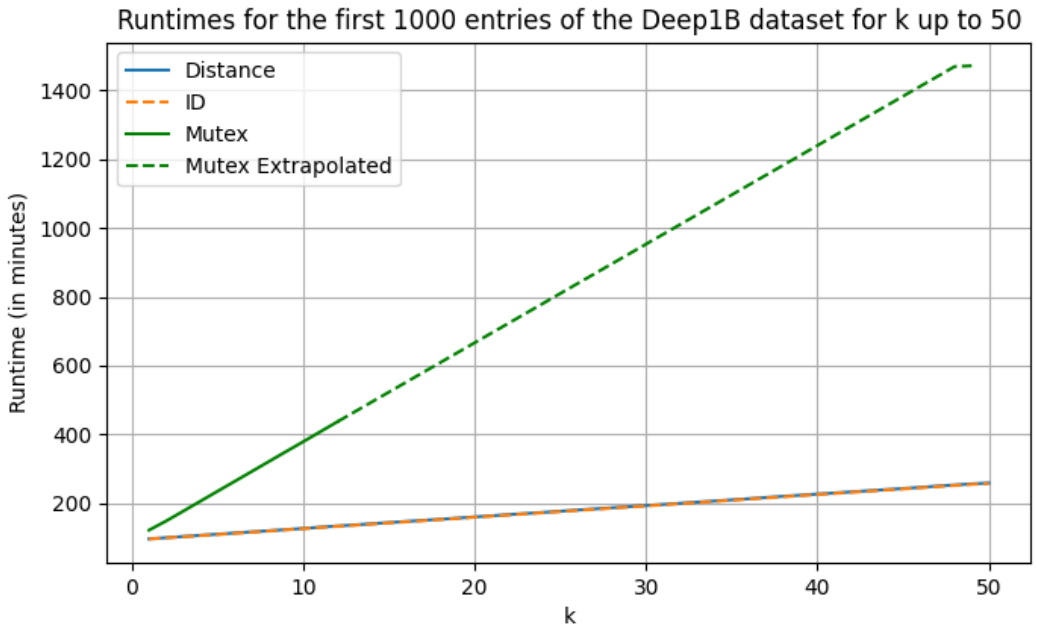


Figure 5.1: Runtime of the first 1000 entries of the Deep1B data set, up to k=50

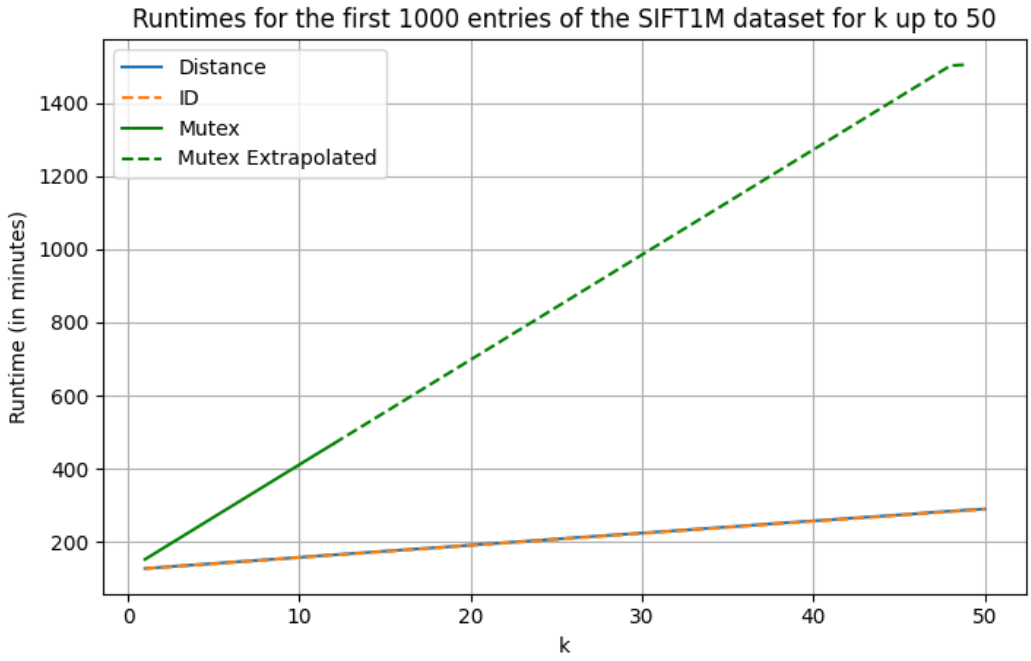


Figure 5.2: Runtime of the first 1000 entries of the SIFT1M data set, up to k=50

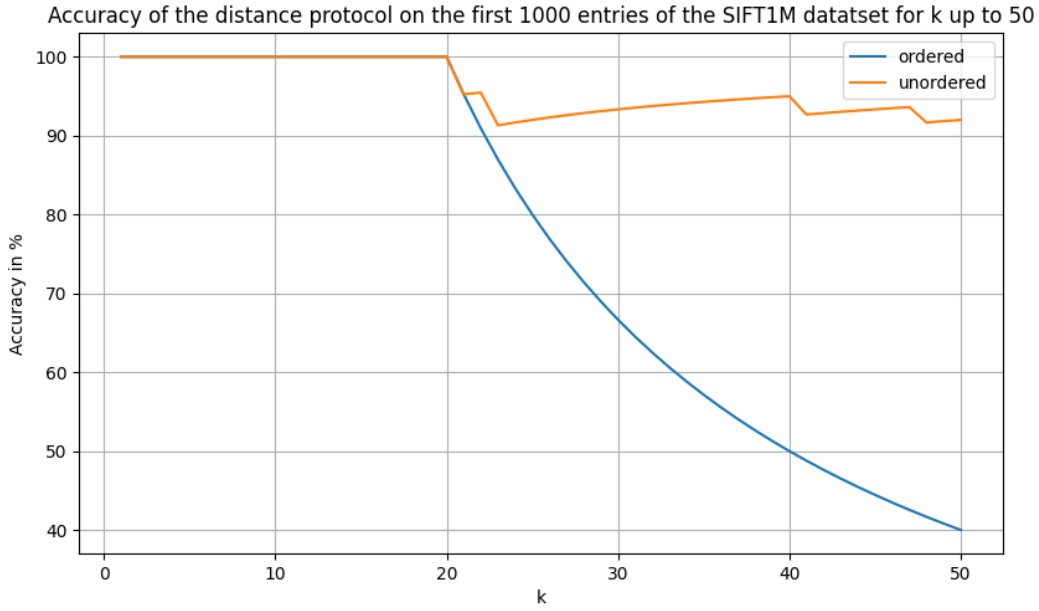


Figure 5.3: Accuracy graph of the first 1000 entries of the SIFT1M data set, up to k=50

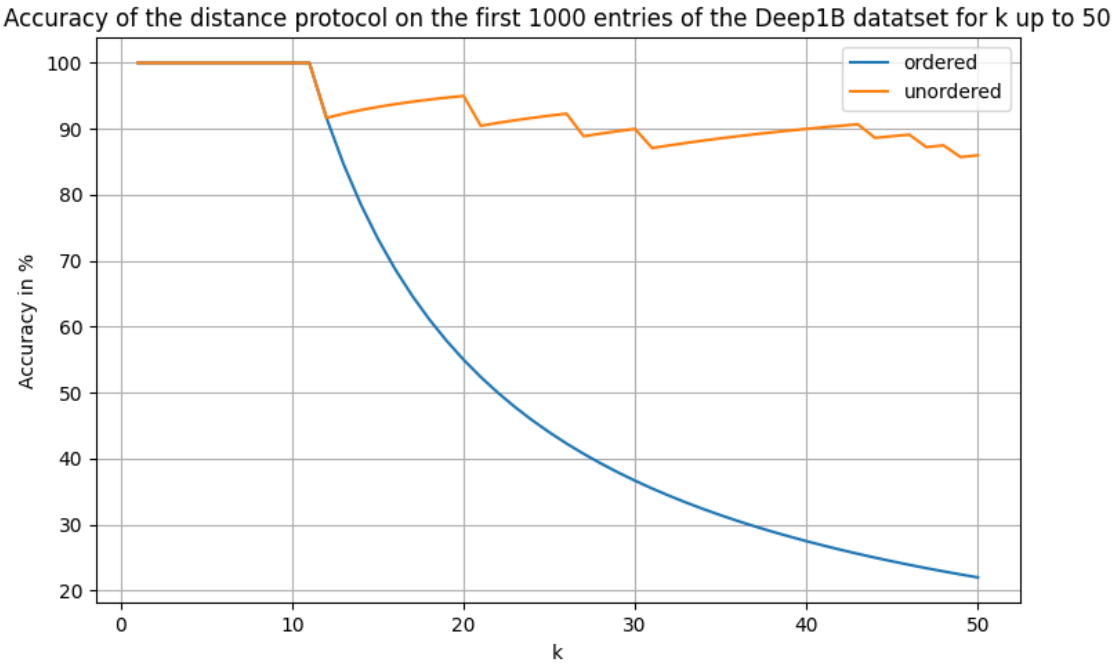


Figure 5.4: Accuracy graph of the first 1000 entries of the Deep1B data set, up to k=50

6

Discussion & Future Work

This chapter discusses our work and compares it to other works, as well as answering the research questions brought up in Chapter 1. Besides that, limitations and future work of our research are discussed, followed by concluding remarks.

6.1. Discussion

6.1.1. Research Questions

To answer our main research question, "How can we perform accurate Privacy Preserving k-nearest neighbour search in outsourced environments using one cloud server and no interaction?", our four sub-questions will be used to answer our main research question.

1. How can we guarantee the privacy of the data and the query in outsourced environments, while still maintaining accurate results?

Our protocol is able to guarantee the privacy of the data and the query by making use of Fully Homomorphic Encryption (FHE), which enables to perform any arbitrary function by performing operations on the ciphertext. By encrypting the data and the query before outsourcing it, it is impossible for the cloud server or any third-parties to learn information about the underlying data without the private key used to decrypt the data. Since FHE makes it possible to perform any arbitrary function, it is possible to gain results similar to plaintext implementations. To guarantee that the results are accurate, the plaintext size for the protocol should be chosen such that it is larger than the sum of the absolute difference between a record and a query. When working with floating point numbers, it is possible to gain the desired accuracy by increasing the plaintext size and scaling the values.

2. How can we construct k-NNS to use only one cloud server?

By using FHE it is possible to perform any arbitrary function by a single party. By sharing a fully homomorphic encrypted database with the cloud service provider, together with the public key, for encrypting, and the server key for performing operations, it allows a client to send queries. The queries should be fully homomorphic encrypted as well, and the results can be decrypted by the client using the private key. Therefore, using the FHE approach, no other parties need to be involved in the protocol.

3. How can we construct k-NNS such that the client is not involved in the calculation?

Again, using FHE the client does not need to be involved in the computation of the protocol, which allows the client to be 'offline', while the cloud service provider does all the calculations. The client is only responsible for sending an encrypted query and decrypting the result.

4. How can we make use of the resources available in outsourced environments

Cloud service providers offer services which allows organizations to rent powerful server instances with more memory and cores than regular computers, without the need for any expertise in cloud computing. Our protocol focuses on making the most use of the large number of cores available in cloud environments. Although TFHE supports parallelization of individual operations, we have shown that parallelizing over larger parts of the computation is significantly faster. The performs difference when parallelizing is possible is clearly visible in Table 5.3 and 5.4 in Chapter 5. When comparing the ID and Distance protocol with the Mutex protocol, where it was not possible to parallelize when updating the distances, only individual operations, the runtime increases from hours to a day.

6.1.2. Comparison to Related Work

Our work is the first Privacy Preserving k-Nearest Neighbour Search protocol to our knowledge, which does not involve non-colluding servers or involve the client in the computation. Our protocol shows it is possible to fully outsource the computation of the k-NNS algorithm to a cloud service provider, while not revealing anything about the underlying ciphertexts.

Compared to the trivial implementation, FHE sorting using bitonic sort or odd-even sort and returning the first k entries, our protocol has a better runtime complexity. The distance computation runtime complexity is the same for sorting and our work. However, recall from Table 4.5, that the best FHE sorting has a runtime complexity of $\mathcal{O}(n(\log_2(n))^2)$, while all three of our protocols have a runtime complexity of $\mathcal{O}(n * k)$ for top-k selection. Therefore, our work outperforms the best known sorting algorithm when $n * k < n(\log_2(n))^2)$.

One of the earliest works done on Privacy Preserving k-Nearest Neighbour Search in outsourced environments, was done by Eldmehdwi et al. [23]. Eldmehdwi et al. proposed a solution where the client is involved with the computation and was able to execute a query on their most secure protocol with $k = 25$, $d = 6$ and $n = 2000$ in roughly 680 minutes, unparrallalized. Although the number of entries in the experiments of Eldmehdwi et al. is twice as large as in our experiments, the size of d is significantly smaller, on which their protocol scales linearly with. The implementation of Eldmehdwi et al. could be parrallelized, however this would mean that the client should also have access to parallelization and thus increasing the computational resources for the client. In our protocol, we are able to perform a query with $k = 25$, $d = 96$ and $m = 1000$ in roughly 190 minutes as seen in Figure 5.1 using 96 threads, in the standalone server scenario.

Further research on the protocol by Eldmehdwi et al. was done by Kim et al. [43], which were able to speed up the protocol up to 24 times, which is only possible when $n >> 2^k$ due to the use of k-d trees. However, the protocol by Kim et al. still required the client to help in the computation, requiring computational and bandwidth resources.

The work by Kesarwani et al. [42] does not require the client to be online during the computation, but rather make use of non-colluding servers. By assuming that two servers are not malicious and do not deviate from the protocol or share any other information, Kesarwani et al. are able to speed up computation in comparison to our work. Kesarwani et al. perform a query with $k = 20$, $d = 32$ and $n = 858$ in under 7 minutes, while our work requires roughly 175 minutes with $k = 20$, $d = 96$ and $n = 1000$. Although the work by Kesarwani et al. is significantly faster, it does require non-colluding servers which our work does not require.

Chen et al. [14] proposed SANNS, a secure approximate nearest neighbour search protocol, which was one of the first works to perform Privacy Preserving k-NNS on large data sets such as SIFT1M and Deep1B in seconds. Chen et al. were able to retrieve the $k = 10$ closest values to a query for the SIFT1M data set with $n = 1.000.000$ and $d = 128$ on their ”slow” network taking 139 seconds and 4.51 GB of bandwidth for the linear implementation, while their clustered implementation took 59.7 seconds and 1.77 GB of bandwidth with a minimum of 90% accuracy. However, the protocol requires the client help in the computation, as well outsourcing the dataset in plaintext. Besides that, the clustering protocol requires to first train a clustering model before it can be used. Although our protocol is significantly slower, it does not require training a clustering model, but only requires encrypting the initial database and sending it to the server and the client is not required to help during the protocol.

6.2. Limitations

A limitation that was briefly discussed, is that the size of the data set used in the ID protocol is limited by the size of the plaintext domain chosen. Although the Distance and Mutex protocol are offered as

Table 6.1: Overview of related work, compared to our solution

Work	Deterministic	Accurate	Standalone Server	k-NNS	Approach
Anonymization [68][51][48]	✖	✖	✓	✓	Anonymization
Vatsalan and Christen [72]	✖	✖	✓	✓	Bloom filter
Huang et al. [31]	✓	✖	✓	✓	LSH
Eldmehdwi et al [23]	✓	✓	✖	✓	Paillier
Kim et al. [43]	✓	✓	✖	✓	Paillier
Kesarwani et al. [43]	✓	✓	✖	✓	BGV
Zheng et al. [77]	?	✖	✖	✓	MASPE
Chen et al. [14]	✓	✖	✖	✓	Garbled Circuits
Shaul et al. [63]	✓	✖	✓	✖	BGV
Jäschke and Armknecht [38]	✓	✓/✖	✓	✖	TFHE
Our work	✓	✓	✓	✓	TFHE

alternatives, they do have their disadvantages, such as sacrificing accuracy or parallelization. The ID protocol can be used in any circumstance, by incrementing the plaintext size used, but doing so increases the computation time.

Besides that, our protocol only supports integers and not any other types such as strings or floats. Floats can be converted into integers by scaling, as done by the Deep1B data set in our experiments. However, doing so loses some accuracy and requires a larger plaintext size to achieve a higher accuracy.

Another limitation is the inability to detect overflows when computing the distances, since the server cannot distinguish between an overflowed ciphertext and a correct one. When the distance overflows, it is possible that the value that is most far away from the query, now is closest to the query and thus an incorrect value is obtained. A solution to solve overflows, is either use a sufficiently large plaintext size, or add additional circuits such that the client is able to see if the results are correct and did not overflow.

Next, in the current implementation it is impossible to parallelize the Mutex protocol due to the protocol deadlocking. The protocol likely deadlocked because of work stealing in the parallelization library used in the TFHE library used. Work stealing allows threads to steal work from other threads if that thread is idle. When the thread who has the lock starts computing, the library detects that other threads are idle and uses the idle threads to compute an operation. However, the current task is now assigned to the idle thread which is waiting for the lock, which the later scheduled lock has and therefore resulting in a deadlock.

On top of that, it might be possible for a malicious client to recover parts of the data set by sending repeated queries. By sending repeated target queries it has been shown that for low-dimensional data it is possible to recover parts of the data sets [45]. Therefore, it might be necessary to limit the number of queries or introduce some noise to the results to prevent leakage.

Finally, speeding up a Privacy Preserving k-NNS does require a significant amount of computing power for a single query, which is not cheap when renting from a cloud service provider. In our experiments, the c6ax24large EC2 instances from AWS were used, having 96 vCPUs, costing $4.6 per hour. So performing one query on a subset of the SIFT1M data set requires around 290 minutes, which costs around $22.23.

6.3. Future Work

During the duration of this research, a new version of the TFHE library, v0.3, has been released, which includes new features which can speed up the computation of our protocol. Changes include faster scalar multiplications, additions, subtractions, as well as a build in CMUX operation and parrallelizable bootstrapping. Although the authors claim faster operations, further research is necessary to determine the impact on our work.

Besides that, our research used the Manhatten distance to use as a distance metric, however numerous other distance metrics exists and are in use as well. Therefore, further research should be done on how to efficiently implement other distance metrics as well.

TFHE was chosen for our protocol due to fast non-linear operations, such as comparisons, which are necessary for top-k selection. Although TFHE is fast for non-linear operations, additions and subtractions are slower than other fully homomorphic encryption schemes. The distance computation runtime shown in Table 5.4 and 5.3 clearly shows that computing distances takes more than a third of the total runtime, which will only increase for a lower k or larger datasets. Recently, advancements in other Fully Homomorphic Encryption schemes, such as BGV, allows performing only slightly slower comparisons than TFHE [32] Therefore, further research in other FHE schemes with fast comparisons and fast arithmetic operations, such as additions and multiplications, can lead to significant improvements of our protocol.

Finally, other non-cryptographic approaches such as Locality Sensitive Hashing (LSH) shows promise in performing Privacy Preserving k-NNS. LSH is especially effective when working with high dimensional data, where our approach scales linearly the size of the dimensions.

6.4. Concluding Remarks

In conclusion, our work shows that it is possible with advances in fully homomorphic encryption, to perform Privacy Preserving k-NNS in the standalone server scenario. Some works focused on assumptions such as that the client is required to have access to computational and bandwidth resources, and help in the computation. Other works used the assumption of non-colluding servers, which does not require the client to interact in the protocol, but uses two non-malicious servers to perform all the computation. Although our work is a multitude of factors slower, our solution does not require any precomputations, interactions or non-colluding assumptions.

We proposed three different protocols which can be used in different scenarios, depending on the size of the database, accuracy and parallelization possible. A subset of the SIFT1M and the Deep1B data sets were used to test the performance of our protocol and showed promising results, being able to compute the top 50 in hours using AWS EC2 instances.

Our protocol hopes to address the discrepancy between having privacy and utility when outsourcing computational tasks to cloud service providers. By using FHE we were able to hide the underlying data, while still gaining accurate results, at the cost of a longer computation time. However, data sets more than a million entries seems unfeasible to use, due to the time and costs of cloud service providers.

Bibliography

[1] Charu C Aggarwal, Alexander Hinneburg, and Daniel A Keim. “On the surprising behavior of distance metrics in high dimensional space”. In: *Database Theory—ICDT 2001: 8th International Conference London, UK, January 4–6, 2001 Proceedings 8*. Springer. 2001, pp. 420–434.

[2] Amazon. *Cloud computing services - Amazon Web Services (AWS)*. 2023. URL: https://aws.amazon.com/.

[3] A. A. Anaj. *KNnClassification.svg - Wikimedia Commons*. May 2007. URL: https://commons.wikimedia.org/w/index.php?curid=2170282.

[4] Louie Andre. *53 important statistics about how much data is created every day*. Jan. 2023. URL: https://financesonline.com/how-much-data-is-created-every-day/.

[5] Artem Babenko and Victor Lempitsky. “Efficient indexing of billion-scale datasets of deep descriptors”. In: *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*. 2016, pp. 2055–2063.

[6] Tony Baer. “Big Data 2019: Cloud redefines the database and Machine Learning runs it”. In: (Jan. 2019). URL: https://www.zdnet.com/article/big-data-2019-cloud-redefines-the-database-and-machine-learning-runs-it/.

[7] Husam Barham. “Achieving competitive advantage through big data: A literature review”. In: *2017 Portland international conference on management of engineering and technology (PICMET)*. IEEE. 2017, pp. 1–7.

[8] Elaine Barker. *Recommendation for key management: Part 1 – general*. May 2020. URL: https://csrc.nist.gov/publications/detail/sp/800-57-part-1/rev-5/final.

[9] Tiago Bianchi. *Google: Advertising revenue 2022*. Feb. 2023. URL: https://www.statista.com/statistics/266249/advertising-revenue-of-google/.

[10] Zvika Brakerski, Craig Gentry, and Vinod Vaikuntanathan. “(Leveled) fully homomorphic encryption without bootstrapping”. In: *ACM Transactions on Computation Theory (TOCT)* 6.3 (2014), pp. 1–36.

[11] Zvika Brakerski et al. “Classical hardness of learning with errors”. In: *Proceedings of the forty-fifth annual ACM symposium on Theory of computing*. 2013, pp. 575–584.

[12] Erik Brynjolfsson and Kristina McElheran. “The rapid adoption of data-driven decision-making”. In: *American Economic Review* 106.5 (2016), pp. 133–139.

[13] Ning Cao et al. “Privacy-preserving multi-keyword ranked search over encrypted cloud data”. In: *IEEE Transactions on parallel and distributed systems* 25.1 (2013), pp. 222–233.

[14] Hao Chen et al. “Sanns: Scaling up secure approximate k-nearest neighbors search”. In: *Proceedings of the 29th USENIX Conference on Security Symposium*. 2020, pp. 2111–2128.

[15] Jung Hee Cheon and Damien Stehlé. “Fully homomophic encryption over the integers revisited”. In: *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer. 2015, pp. 513–536.

[16] Jung Hee Cheon et al. “Homomorphic encryption for arithmetic of approximate numbers”. In: *Advances in Cryptology–ASIACRYPT 2017: 23rd International Conference on the Theory and Applications of Cryptology and Information Security, Hong Kong, China, December 3-7, 2017, Proceedings, Part I 23*. Springer. 2017, pp. 409–437.

[17] Ilaria Chillotti, Marc Joye, and Pascal Paillier. “Programmable bootstrapping enables efficient homomorphic inference of deep neural networks”. In: *Cyber Security Cryptography and Machine Learning: 5th International Symposium, CSCML 2021, Be’er Sheva, Israel, July 8–9, 2021, Proceedings 5*. Springer. 2021, pp. 1–19.

[18] Ilaria Chillotti et al. “TFHE: fast fully homomorphic encryption over the torus”. In: *Journal of Cryptology* 33.1 (2020), pp. 34–91.

[19] Peter Christen et al. “Precise and fast cryptanalysis for Bloom filter based privacy-preserving record linkage”. In: *IEEE Transactions on Knowledge and Data Engineering* 31.11 (2018), pp. 2164–2177.

[20] Google Cloud. *Advantages Of Cloud Computing*. 2022. URL: https://cloud.google.com/learn/advantages-of-cloud-computing.

[21] European Commision. *What personal data is considered sensitive?* 2022. URL: https://commission.europa.eu/law/law-topic/data-protection/reform/rules-business-and-organisations/legal-grounds-processing-data/sensitive-data/what-personal-data-considered-sensitive_en.

[22] Martin Coulter. *Find out if your facebook data was shared with Cambridge Analytica*. Apr. 2018. URL: https://www.standard.co.uk/tech/how-to-find-out-if-your-facebook-data-was-shared-with-cambridge-analytica-using-new-tool-launched-by-the-social-network-today-a3810551.html.

[23] Yousef Elmehdwi, Bharath K Samanthula, and Wei Jiang. “Secure k-nearest neighbor query over encrypted data in outsourced environments”. In: *2014 IEEE 30th International Conference on Data Engineering*. IEEE. 2014, pp. 664–675.

[24] Hilco J van Elten et al. “Big data health care innovations: performance dashboarding as a process of collective sensemaking”. In: *Journal of Medical Internet Research* 24.2 (2022), e30201.

[25] Nitesh Emmadi et al. *Updates on Sorting of Fully Homomorphic Encrypted Data*. Cryptology ePrint Archive, Paper 2015/995. https://eprint.iacr.org/2015/995. 2015. URL: https://eprint.iacr.org/2015/995.

[26] David Eppstein. *bloom filter.svg*. Aug. 2007. URL: https://commons.wikimedia.org/wiki/File:Bloom_filter.svg.

[27] Junfeng Fan and Frederik Vercauteren. “Somewhat practical fully homomorphic encryption”. In: *Cryptology ePrint Archive* (2012).

[28] Maryia Fokina. *Online shopping statistics: Ecommerce trends for 2023*. Jan. 2023. URL: https://www.tidio.com/blog/online-shopping-statistics/.

[29] *General Data Protection Regulation*. Sept. 2022. URL: https://gdpr-info.eu/.

[30] Craig Gentry. *A fully homomorphic encryption scheme*. Stanford university, 2009.

[31] Qiang Huang et al. “Query-aware locality-sensitive hashing for approximate nearest neighbor search”. In: *Proceedings of the VLDB Endowment* 9.1 (2015), pp. 1–12.

[32] Ilia Iliashenko and Vincent Zucca. *Faster homomorphic comparison operations for BGV and BFV*. Cryptology ePrint Archive, Paper 2021/315. https://eprint.iacr.org/2021/315. 2021. URL: https://eprint.iacr.org/2021/315.

[33] Ilia Iliashenko and Vincent Zucca. “Faster homomorphic comparison operations for BGV and BFV”. In: *Proceedings on Privacy Enhancing Technologies* 2021.3 (2021), pp. 246–264.

[34] Piotr Indyk and Rajeev Motwani. “Approximate nearest neighbors: towards removing the curse of dimensionality”. In: *Proceedings of the thirtieth annual ACM symposium on Theory of computing*. 1998, pp. 604–613.

[35] Insightsoftware. *Comparing descriptive, predictive, prescriptive, and diagnostic analytics*. Sept. 2022. URL: https://insightsoftware.com/blog/comparing-descriptive-predictive-prescriptive-and-diagnostic-analytics/.

[36] Jim Isaak and Mina J Hanna. “User data privacy: Facebook, Cambridge Analytica, and privacy protection”. In: *Computer* 51.8 (2018), pp. 56–59.

[37] Angela Jäschke and Frederik Armknecht. “(Finite) field work: Choosing the best encoding of numbers for FHE computation”. In: *Cryptology and Network Security: 16th International Conference, CANS 2017, Hong Kong, China, November 30—December 2, 2017, Revised Selected Papers 16*. Springer. 2018, pp. 482–492.

[38] Angela Jäschke and Frederik Armknecht. “Unsupervised machine learning on encrypted data”. In: *International conference on selected areas in cryptography*. Springer. 2018, pp. 453–478.

[39] Herve Jegou, Matthijs Douze, and Cordelia Schmid. “Product quantization for nearest neighbor search”. In: *IEEE transactions on pattern analysis and machine intelligence* 33.1 (2010), pp. 117–128.

[40] Guojun Ji, Limei Hu, and Kim Hua Tan. “A study on decision-making of food supply chain based on big data”. In: *Journal of systems science and systems engineering* 26 (2017), pp. 183–198.

[41] Batya Kenig and Tamir Tassa. “A practical approximation algorithm for optimal k-anonymity”. In: *Data Mining and Knowledge Discovery* 25 (2012), pp. 134–168.

[42] Manish Kesarwani et al. “Efficient Secure k-Nearest Neighbours over Encrypted Data.” In: *EDBT*. Vol. 2018. 2018, pp. 564–575.

[43] Hyeong-Il Kim, Hyeong-Jin Kim, and Jae-Woo Chang. “A secure kNN query processing algorithm using homomorphic encryption on outsourced database”. In: *Data & knowledge engineering* 123 (2019), p. 101602.

[44] Jakub Klemsa and Melek Önen. “PARMESAN: Parallel ARithMEticS over ENcrypted data”. In: *Cryptology ePrint Archive* (2023).

[45] Evgenios M Kornaropoulos, Charalampos Papamanthou, and Roberto Tamassia. “Data recovery on encrypted databases with k-nearest neighbor query leakage”. In: *2019 IEEE Symposium on Security and Privacy (SP)*. IEEE. 2019, pp. 1033–1050.

[46] Krelixer. *Big Data: Data integrity vs Data Quality*. Feb. 2022. URL: https://krelixir.com/big-data-data-integrity-vs-data-quality/.

[47] Mu-Hsing Kuo et al. “Opportunities and challenges of cloud computing to improve health care services”. In: *Journal of medical Internet research* 13.3 (2011), e1867.

[48] Ninghui Li, Tiancheng Li, and Suresh Venkatasubramanian. “t-closeness: Privacy beyond k-anonymity and l-diversity”. In: *2007 IEEE 23rd international conference on data engineering*. IEEE. 2006, pp. 106–115.

[49] Weipeng Lin et al. “Revisiting security risks of asymmetric scalar product preserving encryption and its variants”. In: *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*. IEEE. 2017, pp. 1116–1125.

[50] Yehuda Lindell. “How to simulate it–a tutorial on the simulation proof technique”. In: *Tutorials on the Foundations of Cryptography: Dedicated to Oded Goldreich* (2017), pp. 277–346.

[51] Ashwin Machanavajjhala et al. “l-diversity: Privacy beyond k-anonymity”. In: *ACM Transactions on Knowledge Discovery from Data (TKDD)* 1.1 (2007), 3–es.

[52] Christian Matt and Ueli Maurer. “The one-time pad revisited”. In: *2013 IEEE International Symposium on Information Theory*. IEEE. 2013, pp. 2706–2710.

[53] Kelsey Miller. *Data-Driven Decision making: A Primer for beginners*. Apr. 2021. URL: https://www.northeastern.edu/graduate/blog/data-driven-decision-making/.

[54] Pascal Paillier. “Public-key cryptosystems based on composite degree residuosity classes”. In: *Advances in Cryptology—EUROCRYPT’99: International Conference on the Theory and Application of Cryptographic Techniques Prague, Czech Republic, May 2–6, 1999 Proceedings 18*. Springer. 1999, pp. 223–238.

[55] Michael S Paterson and Larry J Stockmeyer. “On the number of nonscalar multiplications necessary to evaluate polynomials”. In: *SIAM Journal on Computing* 2.1 (1973), pp. 60–66.

[56] Alex Pentland. “The Data-Driven Society”. In: (Oct. 2013). URL: https://connection.mit.edu/sites/default/files/publication-pdfs/data%20driven%20society%20sci%20amer_0.pdf.

[57] A. Rayaprolu. *25+ impressive big data statistics for 2023*. July 2023. URL: https://techjury.net/blog/big-data-statistics/.

[58] Oded Regev. “On lattices, learning with errors, random linear codes, and cryptography”. In: *Journal of the ACM (JACM)* 56.6 (2009), pp. 1–40.

[59] Allied Market Research. *Big Data Analytics in healthcare market worth 67.82 bn by 2025, says Amr*. May 2020. URL: https://www.globenewswire.com/news-release/2020/05/07/2029608/0/en/Big-Data-Analytics-in-Healthcare-Market-Worth-67-82-Bn-by-2025-Says-AMR.html.

[60] Matthew Rosenberg, Nicholas Confessore, and Carole Cadwalladr. *How trump consultants exploited the Facebook data of Millions*. Mar. 2018. URL: https://www.nytimes.com/2018/03/17/us/politics/cambridge-analytica-trump-campaign.html.

[61] Max Roser, Hannah Ritchie, and Edouard Mathieu. *Technological change*. May 2013. URL: https://ourworldindata.org/technological-change.

[62] Alyssa Schroer and Matthew Urwin. *20 Big Data in healthcare examples and applications*. Feb. 2023. URL: https://builtin.com/big-data/big-data-in-healthcare.

[63] Hayim Shaul, Dan Feldman, and Daniela Rus. "Secure k-ish Nearest Neighbors Classifier". In: *arXiv preprint arXiv:1801.07301* (2018).

[64] Nigel P Smart and Frederik Vercauteren. "Fully homomorphic SIMD operations". In: *Designs, codes and cryptography* 71 (2014), pp. 57–81.

[65] A Smith and Y Zhang. *Canalys Newsroom - Worldwide cloud service spend to grow by 23% in 2023*. Feb. 2023. URL: https://www.canalys.com/newsroom/global-cloud-services-Q4-2022.

[66] the Premerger Notification Office Staff and Stephanie T. Nguyen. *FTC imposes $5 billion penalty and sweeping new privacy restrictions on Facebook*. Jan. 2022. URL: https://www.ftc.gov/news-events/news/press-releases/2019/07/ftc-imposes-5-billion-penalty-sweeping-new-privacy-restrictions-facebook.

[67] T. Stobierski. *The Advantages of Data-Driven Decision-Making*. Aug. 2019. URL: https://online.hbs.edu/blog/post/data-driven-decision-making.

[68] Latanya Sweeney. "k-anonymity: A model for protecting privacy". In: *International journal of uncertainty, fuzziness and knowledge-based systems* 10.05 (2002), pp. 557–570.

[69] TheDigitalArtist. *Download AI Generated Code Matrix Royalty-Free Stock Illustration Image*. June 2023. URL: https://pixabay.com/illustrations/ai-generated-code-matrix-data-8061337/.

[70] Gamze Tillem. "Preserving Confidentiality in Data Analytics-as-a-Service". English. PhD thesis. Delft University of Technology, 2020. ISBN: 978-94-028-2044-7. DOI: 10.4233/uuid:2332e125-c9c4-443c-9a18-0f8fd1c2f85e.

[71] European Union. *Digital economy and society statistics - households and individuals*. Dec. 2022. URL: https://ec.europa.eu/eurostat/statistics-explained/index.php?title=Digital_economy_and_society_statistics_-_households_and_individuals.

[72] Dinusha Vatsalan and Peter Christen. "Privacy-preserving matching of similar patients". In: *Journal of biomedical informatics* 59 (2016), pp. 285–298.

[73] Bing Wang et al. "Privacy-preserving multi-keyword fuzzy search over encrypted data in the cloud". In: *IEEE INFOCOM 2014-IEEE conference on computer communications*. IEEE. 2014, pp. 2112–2120.

[74] Zhifeng Xiao and Yang Xiao. "Security and privacy in cloud computing". In: *IEEE communications surveys & tutorials* 15.2 (2012), pp. 843–859.

[75] Zama. *Security and Cryptography - TFHE-rs_2023*. https://docs.zama.ai/tfhe-rs/getting-started/security_and_cryptography. 2023.

[76] Zama. *TFHE-rs: A Pure Rust Implementation of the TFHE Scheme for Boolean and Integer Arithmetics Over Encrypted Data*. https://github.com/zama-ai/tfhe-rs. 2022.

[77] Yandong Zheng, Rongxing Lu, and Songnian Zhang. "Achieving privacy-preserving weighted similarity range query over outsourced ehealthcare data". In: *ICC 2022-IEEE International Conference on Communications*. IEEE. 2022, pp. 1251–1256.

[78] Yandong Zheng et al. "Achieving efficient and privacy-preserving set containment search over encrypted data". In: *IEEE Transactions on Services Computing* 15.5 (2021), pp. 2604–2618.