A Work Project, presented as part of the requirements for the Award of a Master's degree in Business Analytics from the Nova School of Business and Economics.

Feasibility Study of Data Analysis with Homomorphic Encryption

MORITZ LILLEHOLT HÄCKEL

49558

Work project carried out under the supervision of:

Nuno Preguica

16-12-2022

Table of Contents

Abstract

In this work I examined the existing state-of-the-art methods for privacy preservation in data analytics, and then conducted an experiment to determine the feasibility of one of those techniques, homomorphic encryption, with data analysis on real-life data. The findings of this experiment were that homomorphic encryption is currently not feasible to use in data analytics, but a methodology that is very promising and could help keep data private in the future if certain shortcomings are addressed.

Keywords: data analytics, encryption, privacy, homomorphic encryption, security, differential privacy, k-anonimity, secure multi-party computations

1 Introduction

The AOL queries debacle of 2006 serves as a cautionary tale about the importance of privacy in data analysis. In this incident, a researcher at AOL released a dataset containing search queries from 650,000 users, without properly anonymizing the data. As a result, the personal information of many AOL users was exposed, leading to widespread concern about the protection of privacy in data analysis (Solove 2006). This incident highlights the need for effective privacy preserving techniques in data analysis. In this work, I will first create an overview of existing state-of-the-art techniques for privacy preservation, and then conduct a feasibility study on the use of homomorphic encryption for data analysis, with a particular focus on its application to medical data.

Our study will begin with a literature review of existing techniques and approaches for privacy preserving data analysis. We will focus on four key methods: differential privacy, k-anonymity, homomorphic encryption, and secure multiparty computation. Each of these methods has its own strengths and weaknesses, and we will explore these in detail in order to provide a comprehensive overview of the current state of the art. After our literature review, we will conduct a feasibility study on the use of homomorphic encryption for data analysis. Homomorphic encryption is a relatively new technique that allows data to be encrypted in such a way that it can be analyzed without first being decrypted. This has the potential to provide a high level of privacy protection, as the data remains encrypted throughout the analysis process. We will apply this technique to real-life medical data in order to assess its effectiveness and potential limitations.

2 Literature Review

As data privacy has been an important topic for quite a while, there are already several techniques aiming to keep data anonymous for publishing aggregate information or raw data sets. The following part will examine the existing literature on these techniques and showcase some of the applications they are best suited for.

2.1 Differential Privacy

2.1.1 Methodology

Querying databases that contain sensitive information can often be very important for statistical analysis of relevanta topics such as demographics or medical treatments. One might think that only through aggregate information of querying a database no sensitive information can be derived, but as Dinur and Nissim have shown in their work, it is possible to reconstruct a lot of information within a random level of accuracy using only a finite series of queries (Dinur and Nissim 2003). To avoid such information leakage, Dwork and others have developed a mechanism that aims to protect individuals' information while allowing querying of a database. This technique is called differential privacy.

Differential privacy aims to protect the private data of individuals, so that adding or removing single database items does not allow anyone to draw conclusions about the information of a single person. The overall goal is that results of analyses should not be considerably affected by addition or removal of a single database entry, thereby granting differential privacy (Dwork 2008). To achieve this goal a certain amount of noise, usually in form of a Laplace distribution, is added to the aggregate results. As there is a certain tradeoff between privacy loss and accuracy, where the parameter $\boldsymbol{\varepsilon}$ is used as a privacy loss parameter. This parameter is determines how much noise is added to the result, defining the compromise between accuracy and privacy loss (Wood et al. 2018).

The definition for $\boldsymbol{\varepsilon}$-differential privacy is as follows:

$$Pr[K(D1) \in S] \leq exp(\varepsilon) \times Pr[K(D2) \in S]$$

The randomized function K provides $\boldsymbol{\varepsilon}$-differential privacy if for all databases D1 and D2 differing in at most one element, and $S \subseteq \text{Range}(K)$ the formula above holds true. As long as K adheres to this definition, no person has to be concerned about the leakage of any personal information, since addition or removal of their records does not change the probability of the outputs significantly (Dwork 2008).

The previously mentioned Laplace mechanism for adding numerical noise involves generating random values from a Laplace distribution and adding them to the sensitive data value. The distribution is defined by two parameters: the location parameter, which determines the center of the distribution, and the scale parameter, which determines the spread of the distribution. In the context of the Laplace mechanism, the location parameter is typically set to the sensitive data value itself, and the scale parameter is determined by the privacy parameter (Fernandes, McIver, and Morgan 2021).

If one is to work with strings, categories or trees, instead of adding numerical noise, another method has been thought of called the exponential mechanism. It can be explained through the example of a coin toss. For a Boolean, if the coin shows heads, the real value of the entry is stored, but if the coin shows tails, the coin is flipped again, where heads represent True, and tails represents False. This gives plausible deniability to the individuals in the database.

2.1.2 Application

As of today differential privacy is already being applied to several real life problems. Google was one of the first big tech companies to experiment with differential privacy in different

settings. The first being the collection of user data on Chrome to identify ways unwanted software is seizing control of users' settings. To collect and analyze this data with appropriate privacy for the end users, Google implemented a randomized aggregatable response system called RAPPOR which is based on randomized responses in a differential privacy setting (Erlingsson, Pihur, and Korolova 2014). To build on this successful project, Google later decided to help tackle the problem of urban traffic congestions by providing local research institutions with noisy data about journeys of individuals collected through their Maps application, where no individuals' journey could be identified due to Google relying on differential privacy for anonymization of the data. These research institutions were then able to predict and detect traffic jams with the same accuracy as using sensors in the streets, making these sensors obsolete and thereby possibly saving cities 50,000 € per year by not needing road sensors anymore (Eland 2015).

Apple would follow suit soon and implement differential privacy to observe and analyze user behavior on Safari and Apple devices to improve their user experience. They used their own local differentially private algorithms to do so, meaning the noise would be added to the data on the local device, so no raw private data would reach their servers, adding an additional security layer to this project ("Learning with Privacy at Scale - Apple Machine Learning Research" 2017).

One last application to be considered here is the US Census Bureau. Using a combination of different publicized aggregations from the Census and commercially available datasets, the Census Bureau was able to reidentify 17% of the US population, exposing what is considered sensitive information of individuals. To counteract this, and uphold peoples trust in the anonymity of Census surveys, the Bureau was one of the first institutions to implement differentially private algorithms to anonymize the data they received (Kenny et al. 2021).

2.2 K-anonymity

2.2.1 Methodology

A common way for entities to anonymize data is to simply remove unique identifiers such as the name, address, or social security number, believing that adversaries cannot retrieve personal information from data anonymized in such a manner. As demonstrated in a study, 87% of the US population has a set of attributes that make them unique, with these attributes being the ZIP-code, gender and birthday (Sweeney 2000). By linking two databases on these characteristics researchers have been able to reidentify individuals in anonymized health records, exposing very sensitive data. These attributes, that in combination can be used to identify a person, are called quasi-identifiers.

The k-anonymity protection model aims to avoid aforementioned linkage attacks and other forms of privacy intrusion, while keeping the data practically useful. A dataset is considered to fulfill k-anonymity if for each entry the values of the quasi-identifiers appear at least k times. The following table aims to better explain the concept of k-anonymity:

Table 1: k-anonymity adhering table with k=2

Tuples	**ZIP**	**Birth**	**Gender**	**Disease**
t1	8557*	199*	m	Cancer
t2	8557*	199*	m	Viral Infection
t3	8557*	199*	f	Viral Infection
t4	8557*	199*	f	Viral Infection
t5	8557*	199*	m	Heart Disease

t6	8907*	199*	m	Cancer
t7	8907*	199*	m	Cancer
t8	8907*	199*	f	Heart Disease
t9	8907*	199*	f	Viral Infection

Table 1 is a self-made illustration for a k-anonymity table, with the quasi-identifiers (QI) being ZIP-code, year of birth and gender, and k=2. For each row the values of the QI of that row appear at least twice in the table. More specifically t1(QI) = t2(QI) = t5(QI), t3(QI) = t4(QI), t6(QI) = t7(QI) and t8(QI) = t9(QI). This makes it impossible for someone to uniquely identify an individual based on the QI, and thereby preventing linkage attacks (Sweeney 2002). Following from this each value in the separate QI appears at least k-times, so in this case twice, meaning no person can be uniquely identified by using the QI, to then link them to another database and thereby revealing their identity. This was done by modifying the data by replacing some parts of the numbers with an asterisk and thereby creating buckets.

2.2.2 Application

Compared to other privacy preserving methods that focus more on publishing aggregated results of data, k-anonymity is designed to be used on datasets that get published. There are many different types of publicly available datasets with sensitive information, such as medical data, account breaches or census data to name a few. These datasets have different purposes, but since they all contain sensitive information, proper anonymization that goes beyond the removal of directly identifiable attributes such as name, address and social security number, is very im-

portant, as we've seen that such simple measures often are not sufficient in protecting the individuals from reidentification (Sweeney 2002). One example worth taking a closer look at is the use of k-anonymization on account breaches. As data breaches happen on a daily basis, several account details such as username/email and passwords are available for people with malicious intent to use. To counter this threat there are websites such as 'haveibeenpwned.com' that collect data from these leaks to then enable users to search for their credentials and see if they have been breached and leaked (Thomas et al. 2019). But these sites were facing the security problem of receiving the hashes of not yet breached passwords, making them a target for adversaries looking for passwords. To avoid this, they implemented k-anonymity, where only the first five characters of the hash are passed from the client, who then download a k-anonymized bucket of hashes beginning with the same five characters. Offline the client then checks which of these hashes matches their actual password hash to examine whether their password has been breached (Ali 2018). This process ensures a lot more privacy through the use of k-anonymity, protecting very sensitive information, while keeping the data useable.

There are also some limitations to consider when working with k-anonymity. Some attacks exist that endanger the anonymity of individuals in tables adhering to k-anonymity, such as unsorted matching attacks, complementary release attacks and temporal attacks (Sweeney 2002). It is possible to avoid these and more attacks, but the better protected the data is, the less usable it gets. Even without attacks, some tables might be disclosing sensitive attributes, even though they are k-anonymous, which is referred to as attribute disclosure. Imagine a table containing medical data, that is k-anonymous for the QI, but every entry in the table suffers from the same disease. Adversaries would not need to be able to uniquely identify a person to conclude his or her disease (Domingo-Ferrer and Torra 2008).

2.3 Homomorphic Encryption

2.3.1 Methodology

With an increase in the amount of data collected and complexity of methods to analyze this data, a need in increased computational power for entities has surfaced. As most companies and institutions do not have the necessary computational power in house, cloud computing, where certain companies provide this power on their servers for others to use, has seen a rapid increase in demand. One problem that comes with this approach though, is that companies dealing with sensitive information are concerned about these external servers hosting their customers' sensitive data. Even though cloud computing services guarantee security and value their customer's privacy, this still presupposes customers to trust in the rightness of their statements, which can be hard sometimes considering how hurtful a leakage of sensitive information could be. Entities could store their data ciphered on the cloud, but that would make it impossible to perform computations or queries on it, and thereby preventing them from leveraging the cloud to its full potential.

One way to address this problem is Homomorphic Encryption. The basic idea behind Homomorphic Encryption is that data owners encrypt their data, to then send it to external servers, where calculations are performed on the encrypted data. These encrypted results then get transferred back to the data owner, who can decrypt the results, with the decrypted results being equal to the results had they been computed on the original data (Fontaine and Galand 2007). In doing so, entities will have outsourced the problem of lacking computational power, while not having revealed any sensitive information to a third party.

Mathematically Homomorphic Encryption can be described as follows: M is the set of plaintext and C the respective sets of ciphertexts, with E being the encryption function. A scheme is considered homomorphic if the function E satisfies

$$\forall m_1, m_2 \in M,\ E(m_1 \odot_M m_2) \leftarrow E(m_1) \odot_C E(m_2)$$

Where operators $\odot_M$ in M and $\odot_C$ in C, and $\leftarrow$ means it can be computed directly without an intermediate decryption (Fontaine and Galand 2007). There are two types of schemes, one where the operators are addition, called additively homomorphic, and the second where the operators are multiplication, called multiplicatively homomorphic (Fontaine and Galand 2007). Besides these partially homomorphic encryption schemes, there is also a lot of work being done on a fully homomorphic encryption scheme (FHE), which would allow all computations that can be performed on plain data to also be performed in a homomorphic way. As of today there is no FHE scheme available though, making this a positive outlook for the future (Armknecht et al. 2015).

2.3.2 Application

There are several scenarios where Homomorphic Encryption (HE) can be useful in today's world. In online advertising for example there exists a discord, as people want to get products recommended that suit their needs but are afraid of their preferences being misused by big corporations. To combat this, different HE schemes have been thought of, like a recommender system that encrypts the recommendations, so the system is not aware of the contents (Armknecht and Strufe 2011). Another scheme to mention here is one that is based on the location of individuals. A mobile device sends the users' location to a provider who then sends customized ads back to the user for shops that are near the user. To prevent the provider from observing the users' behavior the data is homomorphically encrypted and the advertisement comes from a third party (Naehrig, Lauter, and Vaikuntanathan 2011).

Another area to consider for HE is medical data. Medical data is very sensitive and could cause a lot of harm should it fall into the wrong hands, like an insurance provider who discriminates against a customer due to insights into his/her medical data. To avoid this dilemma, but at the same time make use of medical data, it can be homomorphically encrypted, making the user the

owner of the data, while still accessing applications that can monitor different health aspects like blood pressure or heart rate (Naehrig, Lauter, and Vaikuntanathan 2011).

A last case to acknowledge is when businesses have information asymmetry and do not trust each other but want to leverage the other's information. Take a company that has sensitive data and wants to use another company's secret algorithm to gain valuable insights from their data. To avoid the company with the algorithm taking advantage of getting the sensitive data, the first company could homomorphically encrypt their data with a so called circuit private scheme, let it run through the algorithm and then decrypt the results, drawing insights from their data, without fearing the other company taking advantage of their data (Armknecht et al. 2015).

But there are also still some limitations to fully homomorphic encryption (FHE). One is if we are looking at a large number of users of the same system. The provider would need to have a single database for every user, encrypted with the user's public key, which creates issues on the provider side especially if there are many users with large databases. Another limiting factor is that running fully homomorphic encryption schemes with complex algorithms leads to a large computational overhead. This increases runtime and in turn also the cost associated with it (Armknecht et al. 2015).

2.4 Multiparty Computation

2.4.1 Methodology

The last technique addresses a different scenario where several parties have inputs with sensitive information and want to compute a function on these inputs, with each party receiving its distinct output, but no other information (Bogetoft et al. 2009). The goal is that even if a certain amount of the parties participating have malicious intentions, they are not able to corrupt the computation or gain any more insights than their own information. Multiparty Computation (MPC) has two important requirements: correctness and privacy. The first one states that each

party should receive its correct output, while privacy means that nothing should be learned by the parties that is not absolutely essential (Lindell 2020).

MPC can be compared to an external trusted party, that receives the inputs from the participants, to then execute the computation and send each participant their proprietary result. This leaves adversaries with the only option to corrupt their own inputs, but they have no alternative to influence the inputs or outputs of the other parties (Lindell 2020).

Most MPC protocols are based on a method called secret sharing. This means that a secret is distributed among a group in a way that no single party has intelligible information about that secret, but if enough secret shares are combined the secret can be reconstructed. Since most MPC use cases require basic arithmetic calculations like multiplication, addition and comparison of integers, most generic MPC protocols can be used quite efficiently (Bogetoft et al. 2009).

2.4.2 Application

MPC can be applied to several real-world problems, making these operations trustworthy and secure. One area where MPC can create value is benchmarking for companies, where firms have their own sensitive information, but want to gauge how they are doing compared to competitors or other players in their market. These other parties also do not want to display their private data to the others, so using MPC they can input their information and get their output, benchmarking them to the other parties. All of this happens confidentially with no party having to fear leakage of their data, while still gaining important information (Bogetoft et al. 2009).

Another area worth investigating for the usefulness of MPC are auctions. There are several types of auctions, where an MPC scheme can replace the auctioneer, making the process more trustworthy and cheaper. The most popular auction types that can benefit from MPC are sealed-

bid auctions, where contestants enter their bid in a sealed fashion, so nobody can tell the other bidders' valuation and only the winning bid is revealed (Bogetoft et al. 2009).

One specific type of auction to take a closer look at is determining the market clearing price for a good, which is the price per unit at which a product is traded. The sellers specify how much of the good they are willing to sell at certain prices, while the buyers define how much they are willing to buy at certain prices. This information then gets transferred to an auctioneer, who calculates the supply and demand for each price. In doing so the price can be determined where total supply equals total demand, which is the market clearing price. As bids reveal a lot of sensitive information about the parties, there are always security concerns, leading to the auctioneer having to be an external third party that can be trusted. Such a task will often be delegated to consultancy houses to ensure correctness and privacy, but as those are also properties of MPC, using such a scheme would save the hosts of the auction much money, while erasing doubts in the integrity of the third party (Bogetoft et al. 2009).

3. Applied Homomorphic Encryption

As I explained the theoretical concepts of different encryption practices in the previous part, this section intends to investigate the practical feasibility of one of the encryption techniques. To do so I am using real-life data to run different analytics on, while looking at how this impacts the performance.

3.1 Homomorphic Encryption

The technique I chose to investigate is Homomorphic Encryption (HE). The reason for my choice is that HE is still a rather new encryption technique, that is not fully developed as of now, but is still something very promising when it comes to protecting data privacy, while still being able to leverage data. Therefore, I will analyze its feasibility in a data analytics setting to

check if HE could already be used in real-life scenarios, or if HE still needs to mature more to be practical.

As HE is a rather new encryption technique, questions arise when it comes to its feasibility on real-life data, in terms of what functionalities can be properly implemented and what the resulting overhead will be from using HE. Also, how the computational overhead will behave depending on the amount of data being processed is something that needs further investigation.

3.1.1 Use Case Scenarios for Homomorphic Encryption

First, we have to understand in what use case and settings the usage of HE is reasonable and helpful. Generally speaking, HE is used when an entity collects data, and wants to leverage this data, by drawing insights out of it using data analytics or machine learning approaches. These data-collecting entities most of the time lack computational power though, to be able to properly leverage their data. This regularly pushes them to suppliers of cloud computing like AWS and Google Cloud, who offer computing on their powerful servers for companies that cannot or don't want to invest in the necessary on-premise capacities. The problem with taking advantage of these cloud computing services is that there used to be no way around exposing your sensitive data to the big companies behind the service, which is difficult if you don't trust in their integrity and promise not to take advantage of your data. Another concern is that you would have to depend on their security measures to ensure no intruder gains access to your sensitive data. Even though the security measures these companies have in place are very sophisticated, increasing the number of places where your data is stored always increases the risk of leakage. This is where HE comes into play, with the possibility to counteract these threats to the integrity of your data. Never exposing your data as plaintext to external entities, will ensure your data is save, while at the same time still enabling computations to be run on it.

3.1.2 Library Selection

The first step in this experiment was to select a library that supports HE being run in a python environment. There were some other criteria that were taken into consideration when choosing a fitting library. These were mainly: compatibility with my machine, availability of information, community engagement and size and ease of use. After considering several different libraries, my choice ultimately was the Concrete library by Zama because there is plenty of information available on how to set it up and there is a community forum where one can get help from others quickly. Zama has two different implementations of the Concrete library, one called Concrete-Numpy, which allows users to implement several numpy functions in an homomorphically encrypted setting, while the other implementation, Concrete-ML, is a library focused on enabling machine learning to be run on encrypted data, based on Scikit Learn (Zama n.d.). Since I am using relatively simple methods to analyze the data, I decided to use the Concrete-Numpy library, as it allows to build my own functions based on numpy operations.

3.2 Dataset

The dataset we ended up choosing for our experiment was the 2015 de-identified NY inpatient discharge (SPARCS) dataset, provided to the public by New York State Department of Health. This file contains patient characteristics, such as charges, treatments, diagnoses, length of stay and many more, for after they have been discharged from the hospital. All data is de-identified, meaning all identifiable elements have been removed, to protect the patients' identities. This is also the only way the New York State Department of Health is allowed to publish the discharge data for external people to work with. Other data scientists have used this dataset to predict certain features such as the length of stay using machine learning approaches, whereas I will be using it to calculate certain statistics.

3.2.1 Importance of Confidentiality

Besides the public availability of the dataset there were also other reasons for my choice. As I am working with encryption techniques, there needs to be a valid reason to take on the computational overhead that comes with encryption, meaning the data or its subjects need to be protected from adversaries. When it comes to how sensitive personal data is to people, medical data probably ranks in the top spots, considering how intimate this information usually is. There is plenty of harm to be done to an individual, if their medical data falls into the wrong hands such as blackmailing or discrimination from employers and insurance. Also looking at it from an HE standpoint, this data can be very valuable to large corporations, who might want to use it to create profiles of individuals that can be used for advertising or otherwise monetized. That is why we will use HE to transmit encrypted data to third parties, preventing them from taking advantage or making profit of sensitive data, while still being able to perform computations on them, while we are the only ones who can decrypt the results using our key.

3.2.2 Usefulness of Medical Data

Medical data in general is incredibly useful in a data analytics context because it allows researchers and medical professionals to gain valuable insights into the effectiveness of treatments, predict potential health outcomes, and improve patient care. By analyzing large sets of medical data, researchers can identify trends and patterns that can help them better understand the underlying causes of diseases and develop more effective treatments. This information can also be used to improve the efficiency and accuracy of medical diagnoses, as well as to identify potential risk factors for certain conditions. Overall, the ability to analyze medical data using data analytics techniques is a powerful tool that has the potential to greatly improve the quality of healthcare (Itani, Lecron, and Fortemps 2019).

Hospital inpatient discharge data more specifically allows health care experts to gain a better understanding of the most common procedures and treatments that are being performed, as well

as the outcomes of those treatments. This information can then be used to identify areas for improvement in medical care, such as identifying treatments that have a high rate of success or procedures that may be causing complications for patients. Additionally, Hospital Inpatient Discharges data can also be used to monitor trends in the healthcare industry, such as the rise of certain medical conditions or the adoption of new technologies (Schoenman et al. 2007).

Therefore, it is extremely important to be able to take advantage of as much medical data as possible. Encrypting this data allows us to use more data, as patients do not have to worry about their sensitive information being exposed to other parties.

3.3 Experiment

To figure out whether HE is feasible for analyzing the provided data, I am going to run two different analytics, one calculating the percentage of patients that received a certain diagnosis, and the second calculating the average of the length of stay.

3.2.1 Setup

After downloading and deploying the docker image of the Concrete-Numpy library in a Jupyter Notebook, I imported the aforementioned dataset and did some data cleaning to be able to use it properly for the homomorphic calculations. More specifically the Length of Stay column, which will be used in the average calculation was converted to data type int, columns that were of no use or information were dropped and only one quarter of the entire dataset was used, as the entire set was too large and would end up occupying to much memory to be able to run homomorphically encrypted functions properly. Furthermore, each homomorphic function needed to be compiled in the first place, which also included giving it an inputset that represents the typical inputs to the function. Also I included the key generation in the compiling part of the function, as this was something that was computationally very heavy. If one were to not explicitly execute the key generation function beforehand, it will be performed when running

the homomorphic function for the first time, and thereby heavily distorting the runtime comparisons.

One downside of the Concrete-Numpy library is that a new function needs to be compiled for every different size of inputs, meaning I had to compile these functions several times and generate new keys for each new function to be able to compare how the size of the input impacts the overhead.

3.3.2 Distribution

The first calculation I am performing with HE is computing the share of a certain group. In my example I am checking for the percentage of patients in the age group of 18 to 29, that were admitted to the hospital with a condition that was considered as major severity. The function used for this calculation can also be adjusted to look for other severity levels and other sub groups of the dataset. Using HE the input array of APR Severity of Illness Code is handed over to the function, where it is encrypted. Then the numpy where function checks the encrypted values for accordance and outputs an array of 1s where the condition was met, and 0s where the condition was not met. This array is then used as plaintext to calculate the percentage of 1s in relation to the size of the array.

3.3.2.1 Performance Compared to Unencrypted Computation

There were quite some differences in terms of performance when comparing the function running homomorphically encrypted to an unencrypted computation, as was expected. The positive news is that the time it took to compile each homomorphic function was not very long, around 8.6 seconds, and there were almost no differences in relation to the amount of datapoints used as input for compiling the function. Also,

nr_entries	times_calc_plain	times_calc_encr	factor
25.0	0.000961	18.171822	18913.0
40.0	0.000249	22.891470	91791.0
50.0	0.000193	29.466722	152961.0
60.0	0.000165	33.906683	205810.0
75.0	0.000165	42.550858	257163.0
90.0	0.000144	51.111129	353755.0
100.0	0.000130	56.257546	431373.0

Table 2: Comparison of calculation times for the distribution

both functions arrived at the same results, proving that HE will yield correct answers in this setting.

But the difference of computing times of the two was quite significant as was to be expected, as can be seen in figure X. While the HE took from 16 up to 55 seconds to run, depending on the input size, the unencrypted computation was performed in the span of 0.13 and 0.96 milli-seconds. Also, the factor of difference for each calculation increased steadily with increasing input size. This is obvious, when observing that the plain calculation times decreased, while the encrypted ones increase.

3.3.2.2 Performance Depending on Input Size

When looking at how the homomorphically encrypted function performed regarding increasing input size, it can be noted that an increase in inputs also increases the computational time. As very clearly observable in figure 1 the increase in computing time is almost proportional to the number of datapoints handed to the function, resulting in a close to linear growth.

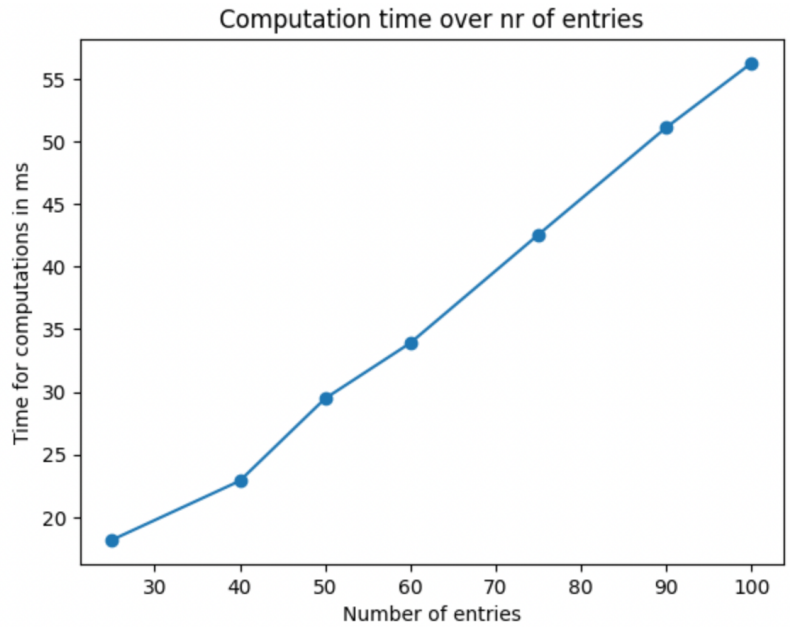

Figure 1: Computation time of distribution over number of entries

This may raise concerns about the feasibility of HE, as most data analytics depend on large sample sizes to make statistically relevant statements. The linear increase might make using this HE function in real-life unfeasible, as large inputs will require a lot of computing to be performed.

3.3.3 Mean

As a second statistic I decided to calculate the mean of the column 'Length of Stay', for patients that stayed at the hospital in the county of Cattaraugus for less than two weeks. Since the numpy mean function is not available in the Concrete Numpy library, I had to calculate the average by taking the sum of the input array and dividing it by its size. Also, I had to rely on a floor_divide, which returns the largest smaller or equal integer to the division of the inputs since the Concrete Numpy library cannot return floats as output.

3.3.3.1 Mean Performance Compared to Unencrypted Computation

The performance factor of comparing the encrypted mean calculation with the plain mean calculation was varying quite strongly, between a factor of ~71000 and ~261000 (see). It seems that this factor increases with increasing input size, even though it will have to be tested if that holds true for even larger inputs. Nontheless, this is still a very significant difference in computing times, which makes it difficult to argue that HE is practical for that kind of computation.

nr_entries	times_calc_plain	times_calc_encr	factor
5.0	0.000244	17.522488	71842.0
10.0	0.000269	21.026711	78116.0
15.0	0.000123	20.881067	170392.0
20.0	0.000145	37.975723	261118.0

Table 3: Comparison of computation times of the mean function

3.3.3.2 Mean Performance Depending on Input Size

The computational increase regarding the input size was not as linear as it was for the first experiment. This time we can observe a sharp increase for the last iteration with an input array of size 20 (see figure 2). Unfortunately, my machine was not able to handle larger inputs, which

would have been useful to clarify if for a larger sample size, the increase in computation also follows a linear form, or another form of growth. Still, this gives doubts whether calculating a mean is something feasible in a homomorphically encrypted setting.

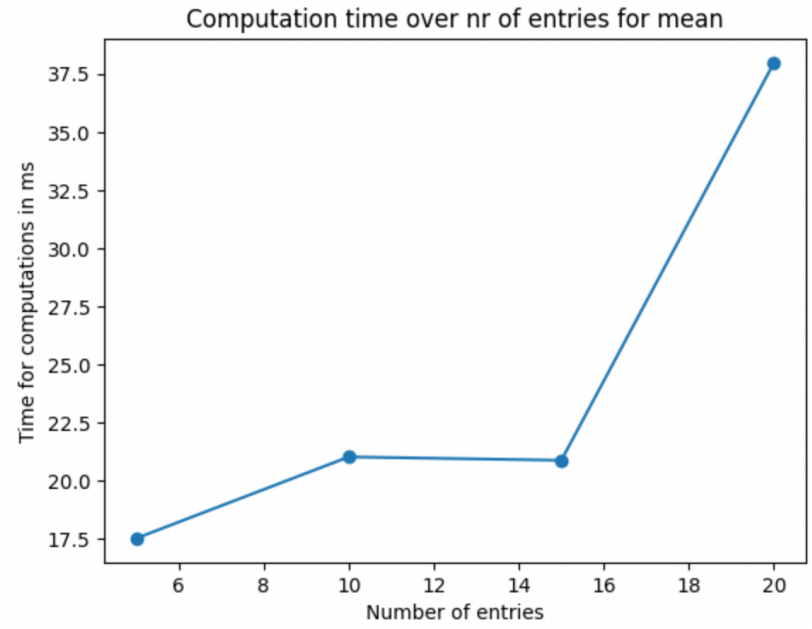

Figure 2: Computation time of the mean function of the number of entries

3.4 Limitations

As of the time of writing this paper, there were still quite a few limitations to the usage of HE in analyzing data sets. One limitation when working with the Concrete-Numpy library was that there is a 7-bit precision limit. This means that the end-user can only input values of a range of 127. At this point in time the company behind the library is working on expanding this limit, but as it is a critical part to making HE run properly, it is not an easy task and even increasing the limit by one bit still limits the user to a range of 255. Especially in a data analytics context, this range is too small to be able to run significant analyses, as most real-life data will exceed this limit. This also limits some computations, for example the sum function can only output results that are inside of the precision limit, once they exceed them the results are incorrect.

Another limitation is the fact, that as of today there is only a limited amount of numpy functions that can be implemented in a homomorphic way in the Concrete-Numpy library. There are still several functions that would be very helpful to conduct data analysis over a dataset that are not available. Some examples include finding the maximum and minimum of an array or calculating the median among others.

The last limitation to the usage of this library in a data analytics scenario, is the computational effort that goes into compiling and generating the keys for the functions in a homomorphic way. One needs to compile a new function for every different size of the input, making it very static and impractical to use on datasets of different sizes. Another large issue that comes with this inflexibility is that each time a new function gets compiled, new keys have to be generated, which is a process that takes a lot of computational effort and the long running times associated with it make the adoption of HE in data analytics difficult. The mean key generation for the mean function with 20 data points took up to 50 minutes in some cases for example.

3.5 Findings

There are some interesting findings to take away from the conducted experiment. First off, the long key generation times that come with the mean function make it very unpractical to use, especially when considering that the function needs to be compiled anew if the size of the input changes. Also, the fact that I had to keep the range of input numbers quite low, and still could only compile the function on a very small set of entries without my machine dying, goes to show that the computation that is necessary for this function makes it unfeasible to use, maybe except for using a very powerful machine for the computation. Furthermore, the significant difference in performance between encrypted and plain calculations was deterring, as there was a factor of between 19000 and 216000, when comparing the two.

Another interesting insight was the development of computing time in relation to the size of the input. There was a linear increase visible in relation to the input size which would have to be investigated further, on whether that growth rate keeps true for larger inputs. This allows for quite precise estimations on how long the calculations will take, but also shows with larger inputs, the computation time grows proportionally, which makes it difficult to argue in favor of the feasibility of HE in data analysis.

Overall, this paper comes to the conclusion that HE is not feasible to use in a data analytics context, as the overhead from using this technique is too large, and the limitations that come with it are too manifold. Especially when considering that data analytics rely on large amounts of data to derive statistically valid and important insights, the proportional increase in computation that comes with these calculations on large data sets makes it unpractical to use.

Other options for anonymizing data currently are better fit for the usage in data analytics. One such example would be the previously discussed differential privacy. In the same setting, the noise could be added to the original data, disguising identifying attributes. This modified data could then be sent to cloud computing services, who could perform calculations, while not having access to the original data. In terms of how much the service provider could get out of the data, HE is more privacy preserving, but in terms of practicability something like differential privacy makes more sense when considering the current status of HE.

3.6 Future Possibilities

There are several future possibilities where HE can be of tremendous assistance to solving problems that exist, if homomorphic encryptions keep improving in terms of usability and efficiency. One of those possibilities depends on what is known as the grail of HE, a fully homomorphic encryption scheme (FHE) which allows arbitrary computations to be performed on encrypted data, meaning computations that can be performed on unencrypted data can also be

performed on encrypted data (Armknecht et al. 2015). This in combination with better computational efficiency could make HE interesting to use in data analytics and several other disciplines.

Another area that could be assisted by HE are distributed computing systems. One of the main challenges with distributed computing systems is that the data is often sensitive or private, and it is not always possible or desirable to share the data with the other parties involved in the computation. Homomorphic encryption could be used to encrypt the data at each party, allowing the parties to perform the computation on the encrypted data without revealing the underlying data to each other (Zhao, Li, and Liu 2014). This could enable distributed computing systems to be implemented in scenarios where the data is sensitive or where the parties do not trust each other, similar or complementary to secure multi-party computations that were discussed in the first part of this paper.

One of the main opportunities for using homomorphic encryption is in combination with machine learning. This allows entities to perform machine learning on sensitive data without revealing the underlying data to the machine learning algorithm. This could be particularly useful in applications where the data contains sensitive information, such as medical records or financial data. By encrypting the data using homomorphic encryption, it would be possible to perform machine learning on the encrypted data without revealing the sensitive information to the machine learning algorithm (Aslett, Esperança, and Holmes 2015). Ensuring that sensitive information remains sensitive would allow machine learning algorithms to be trained on plenty more data that was previously considered out of reach, as it was too sensitive for companies to gain access to, examples for this include medical and financial data.

4. Conclusion

Homomorphic encryption is a powerful tool that allows data analysis to be performed on encrypted data without the need to decrypt it first. This property enables a wide range of applications in various fields, such as secure cloud computing, secure multi-party computation, and privacy-preserving machine learning. However, the feasibility of using HE for data analysis depends on several factors, such as the efficiency and scalability of the underlying algorithms, the security of the encryption scheme, and the availability of suitable hardware.

In my experiment it was found that HE is not a feasible solution for analyzing large amounts of medical data in the set up that I could provide. Maybe more powerful machines will make this more feasible, but for simple data analyses it seems that other encryption methods such as differential privacy have the upper hand in terms of practicability.

But HE is a quite immature technology, that still is worked on by a plethora of people, making it a promising idea for the future. When efficiency and overhead concerns will be addressed, and fully homomorphic schemes are available HE could revolutionize data privacy, while improving data availability for all types of applications, the most interesting being machine learning. This seems to be the direction HE is heading in, focusing more on advanced applications such as machine learning and distributed computing, rather than data analytics, as there are already sufficient techniques for that, that anonymize personal information properly.

In conclusion, HE is a non-feasible approach to secure data analysis, but further research and development are needed to overcome the challenges and limitations of the current state-of-the-art techniques. By addressing these challenges, homomorphic encryption has the potential to become a key enabling technology for secure and privacy-preserving data analysis.

References

Ali, Junade. 2018. “Validating Leaked Passwords with K-Anonymity.” 2018. https://blog.cloudflare.com/validating-leaked-passwords-with-k-anonymity/.

Armknecht, Frederik, Colin Boyd, Christopher Carr, Angela Jaschke, and Christian A Reuter. 2015. “A Guide to Fully Homomorphic Encryption,” 35.

Armknecht, Frederik, and Thorsten Strufe. 2011. “An Efficient Distributed Privacy-Preserving Recommendation System.” In *2011 The 10th IFIP Annual Mediterranean Ad Hoc Networking Workshop*, 65–70. Favignana Island, Sicily, Italy: IEEE. https://doi.org/10.1109/Med-Hoc-Net.2011.5970495.

Aslett, Louis J. M., Pedro M. Esperança, and Chris C. Holmes. 2015. “A Review of Homomorphic Encryption and Software Tools for Encrypted Statistical Machine Learning.” arXiv. http://arxiv.org/abs/1508.06574.

Bogetoft, Peter, Dan Lund Christensen, Ivan Damgård, Martin Geisler, Thomas Jakobsen, Mikkel Krøigaard, Janus Dam Nielsen, et al. 2009. “Secure Multiparty Computation Goes Live.” In *Financial Cryptography and Data Security*, edited by Roger Dingledine and Philippe Golle, 5628:325–43. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-642-03549-4_20.

Dinur, Irit, and Kobbi Nissim. 2003. “Revealing Information While Preserving Privacy.” In *Proceedings of the Twenty-Second ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems - PODS ’03*, 202–10. San Diego, California: ACM Press. https://doi.org/10.1145/773153.773173.

Domingo-Ferrer, Josep, and Vicen Torra. 2008. “A Critique of K-Anonymity and Some of Its Enhancements.” In *2008 Third International Conference on Availability, Reliability and Security*, 990–93. IEEE. https://doi.org/10.1109/ARES.2008.97.

Dwork, Cynthia. 2008. “Differential Privacy: A Survey of Results.” In *Theory and Applications of Models of Computation*, edited by Manindra Agrawal, Dingzhu Du, Zhenhua Duan, and Angsheng Li, 4978:1–19. Lecture Notes in Computer Science. Berlin, Heidelberg: Springer Berlin Heidelberg. https://doi.org/10.1007/978-3-540-79228-4_1.

Eland, Andrew. 2015. “Tackling Urban Mobility with Technology.” *Google Europe Blog* (blog). 2015. https://europe.googleblog.com/2015/11/tackling-urban-mobility-with-technology.html.

Erlingsson, Úlfar, Vasyl Pihur, and Aleksandra Korolova. 2014. “RAPPOR: Randomized Aggregatable Privacy-Preserving Ordinal Response.” In *Proceedings of the 2014 ACM SIGSAC Conference on Computer and Communications Security*, 1054–67. Scottsdale Arizona USA: ACM. https://doi.org/10.1145/2660267.2660348.

Fernandes, Natasha, Annabelle McIver, and Carroll Morgan. 2021. “The Laplace Mechanism Has Optimal Utility for Differential Privacy over Continuous Queries.” In *2021 36th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)*, 1–12. Rome, Italy: IEEE. https://doi.org/10.1109/LICS52264.2021.9470718.

Fontaine, Caroline, and Fabien Galand. 2007. “A Survey of Homomorphic Encryption for Nonspecialists.” *EURASIP Journal on Information Security* 2007: 1–10. https://doi.org/10.1155/2007/13801.

Itani, Sarah, Fabian Lecron, and Philippe Fortemps. 2019. “Specifics of Medical Data Mining for Diagnosis Aid: A Survey.” *Expert Systems with Applications* 118 (March): 300–314. https://doi.org/10.1016/j.eswa.2018.09.056.

Kenny, Christopher T., Shiro Kuriwaki, Cory McCartan, Evan T. R. Rosenman, Tyler Simko, and Kosuke Imai. 2021. “The Use of Differential Privacy for Census Data and Its Impact on Redistricting: The Case of the 2020 U.S. Census.” *Science Advances* 7 (41): eabk3283. https://doi.org/10.1126/sciadv.abk3283.

“Learning with Privacy at Scale - Apple Machine Learning Research.” 2017. December 2017. https://machinelearning.apple.com/research/learning-with-privacy-at-scale.

Lindell, Yehuda. 2020. “Secure Multiparty Computation (MPC),” 15. https://doi.org/10.1145/3387108.

Naehrig, Michael, Kristin Lauter, and Vinod Vaikuntanathan. 2011. “Can Homomorphic Encryption Be Practical?” In *Proceedings of the 3rd ACM Workshop on Cloud Computing Security Workshop - CCSW ’11*, 113. Chicago, Illinois, USA: ACM Press. https://doi.org/10.1145/2046660.2046682.

Schoenman, Julie A., Janet P. Sutton, Anne Elixhauser, and Denise Love. 2007. “Understanding and Enhancing the Value of Hospital Discharge Data.” *Medical Care Research and Review* 64 (4): 449–68. https://doi.org/10.1177/1077558707301963.

Solove, Daniel. 2006. “The AOL Privacy Debacle: Internet Search Queries and Privacy.” TeachPrivacy. August 10, 2006. https://teachprivacy.com/aol-privacy-debacle-internet-search-queries-and-privacy/.

Sweeney, Latanya. 2000. “Simple Demographics Often Identify People Uniquely.” . . *Pittsburgh*, 34.

———. 2002. “K-ANONYMITY: A MODEL FOR PROTECTING PRIVACY.” *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems* 10 (05): 557–70. https://doi.org/10.1142/S0218488502001648.

Thomas, Kurt, Jennifer Pullman, Kevin Yeo, Ananth Raghunathan, Patrick Gage Kelley, Luca Invernizzi, Borbala Benko, et al. 2019. “Protecting Accounts from Credential Stuffing with Password Breach Alerting,” 18.

Wood, Alexandra, Micah Altman, Aaron Bembenek, Mark Bun, Marco Gaboardi, James Honaker, Kobbi Nissim, David O’Brien, Thomas Steinke, and Salil Vadhan. 2018. “Differential Privacy: A Primer for a Non-Technical Audience.” *SSRN Electronic Journal*. https://doi.org/10.2139/ssrn.3338027.

Zama. n.d. “What Is Concrete ML?” Accessed December 7, 2022. https://docs.zama.ai/concrete-ml.

Zhao, Feng, Chao Li, and Chun Feng Liu. 2014. “A Cloud Computing Security Solution Based on Fully Homomorphic Encryption.” In *16th International Conference on Advanced Communication Technology*, 485–88. Pyeongchang, Korea (South): Global IT Research Institute (GIRI). https://doi.org/10.1109/ICACT.2014.6779008.

Appendix

Code

Github: https://github.com/mohaeckel/master_thesis/blob/main/Thesis_homomorphic_encryption.ipynb

GoogleDrive:

https://drive.google.com/drive/folders/1DHjaA30xjGahxNGb3EalX3fK2gBUM3R_?usp=sharing