CHALMERS
UNIVERSITY OF TECHNOLOGY

UNIVERSITY OF GOTHENBURG

Protecting Patient Privacy in Healthcare Analytics with Fully Homomorphic Encryption and Differential Privacy

Master's thesis in Computer science and engineering

SHAHNUR ISGANDARLI
JOAKIM WENNERBERG

Department of Computer Science and Engineering
CHALMERS UNIVERSITY OF TECHNOLOGY
UNIVERSITY OF GOTHENBURG
Gothenburg, Sweden 2026

Master's thesis 2026

Protecting Patient Privacy in Healthcare Analytics with Fully Homomorphic Encryption and Differential Privacy

SHAHNUR ISGANDARLI
JOAKIM WENNERBERG

UNIVERSITY OF
GOTHENBURG

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Computer Science and Engineering
Chalmers University of Technology
University of Gothenburg
Gothenburg, Sweden 2026

Protecting Patient Privacy in Healthcare Analytics with Fully Homomorphic Encryption and Differential Privacy

SHAHNUR ISGANDARLI
JOAKIM WENNERBERG

Supervisor: Rhouma Rhouma, Department of Computer Science and Engineering
Examiner: Ahmed Ali-Eldin Hassan, Department of Computer Science and Engineering

Master's Thesis 2026
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg
SE-412 96 Gothenburg
Telephone +46 31 772 1000

Typeset in LaTeX
Gothenburg, Sweden 2026

Protecting Patient Privacy in Healthcare Analytics with Fully Homomorphic Encryption and Differential Privacy

SHAHNUR ISGANDARLI
JOAKIM WENNERBERG
Department of Computer Science and Engineering
Chalmers University of Technology and University of Gothenburg

Abstract

Data analytics in the healthcare domain requires access to sensitive patient information, creating conflicting interests between the need for usability and privacy requirements. Fully Homomorphic Encryption (FHE) enables computation on encrypted data, while Differential Privacy (DP) protects individuals against inference attacks by introducing controlled noise into aggregate query results.

This thesis investigates the combined use of FHE and DP for privacy-preserving healthcare analytics and evaluates the resulting privacy guarantees, performance, and practical limitations. Three aggregation queries are implemented and evaluated in a multi-party privacy-preserving system using multiple FHE schemes and libraries, including the BFV, BGV, CKKS, and TFHE schemes using the Microsoft SEAL and Concrete FHE libraries. Performance, accuracy, ciphertext expansion, and compliance to confidentiality and availability requirements are assessed using a synthetic healthcare dataset.

The results show that combining FHE and DP strengthens protection against eavesdropping and membership inference attacks compared to using either of the methods alone. However, the increased privacy comes at a large cost in performance and usability. Encrypted query execution is orders of magnitude slower than plaintext execution, and current FHE libraries provide limited support for common statistical operations. Additionally, even in the best case, ciphertexts span several megabytes for a single value, although this can be partially mitigated through compression prior to storage or network transmission. Finally, executing homomorphic computations in an insecure environment could expose encrypted data to side-channel attacks such as power measurement or timing attacks.

These limitations represent a significant hindrance for large-scale deployment of FHE in statistical contexts. Improvements to the FHE ecosystem in regards to performance and usability could enable future large-scale deployment.

Keywords: Cryptography, homomorphic encryption, privacy-preserving system, differential privacy, security

Acknowledgements

We would like to thank our supervisor, Rhouma, for his support during the course of the project. His guidance was essential for the completion of this project. We are grateful for his advice and direction, which proved to be very valuable.

Shahnur Isgandarli & Joakim Wennerberg, Gothenburg, 2026-01-21

Contents

List of Figures

List of Tables

1

Introduction

Data analysis in the healthcare domain typically requires access to the sensitive patient data. However, data privacy regulations impose restrictions on data usage, which, in practice, often prevent direct access to the data. While there are systems that provide data for clinical research, they usually enforce strict regulations on how the data can be used, including how data can be stored and processed. The complexities of data sharing to stakeholders present a significant roadblock to advances in medicine [1].

Traditional methods of de-identifying data typically involve removing personal information from datasets, such as names, birth dates, and other demographic information. However, these approaches are vulnerable to membership inference attacks [2], which aim to determine whether a specific individual is included in the dataset. Since re-identification attacks can still be carried out on de-identified data, removing explicit identifiers alone is insufficient to guarantee privacy. Therefore, strict requirements on data processing remain necessary even when datasets have been de-identified.

This project investigates methods for enabling data analytics on the sensitive patient data without compromising patient privacy. Specifically, it explores two privacy-preserving methods: Fully Homomorphic Encryption (FHE) and Differential Privacy (DP). FHE allows computations to be performed directly on encrypted data, enabling secure processing in environments where an adversary can read the memory during computations. We refer to such environments as *untrusted* in this thesis. DP, on the other hand, introduces noise into query results, which protects private information while still allowing aggregate data analysis. With these methods, we aim to explore possibilities for relaxing current restrictions on where healthcare data can be processed and who can access it.

1.1 Related research

Application of Fully Homomorphic Encryption and Differential Privacy in combination in privacy-preserving systems has previously been researched in the domains of IoT and Federated Learning (FL). Some research has been done on privacy-preserving data analysis, such as [3] and [4]. In this section we aim to distinguish this project from this previous research.

Aziz et al. conducted a survey on the application of privacy-preserving techniques, with focus on DP and FHE in the area of Federated Learning (FL) [5]. They argue that combining the two methods together potentially mitigates the drawbacks created by each method on its own [5]. For DP, the main challenge is choosing a parameter which balances accuracy with privacy, whereas for FHE, the main challenge is balancing privacy with performance [5]. Combining FHE and DP in privacy-preserving systems has an impact on performance, privacy, and accuracy.

Kim et al. explored the application of FHE in text embeddings, and demonstrated a system where a user sends an encrypted query together with a public key to a financial service [4]. The financial service computes a similarity search in an embeddings database, and returns the encrypted index to the user for decryption [4]. However, this scheme assumes that the environment in which homomorphic computations are executed in is a trusted environment. This assumption contradicts the goal of being able to execute the computations in an untrusted environment, since the computation server has access to the private key.

DP and FHE have also previously been applied in healthcare analytics. Raisaro et al. combined these techniques to protect patients genomic data using real patient datasets [3] in a real-life system. In this solution, the differential privacy noise is added by the server that does the homomorphic computations in the result of the query before returning it back to the user [3]. This contradicts the same principle as [4], since the untrusted environment would have access to unperturbed encrypted data, and therefore it would not protect against membership inference attacks if the attacker is able to decrypt the data.

So far, we have not found research that explores the usage of FHE and DP in a scenario where homomorphic computations are done in an untrusted environment in the context of data mining or data analysis. Aside from these two papers, we did not find any papers that implemented FHE and DP in a data analysis scenario. Therefore, we argue that there is a gap in the current research and that this project is both relevant and novel, and presents scientific and engineering challenges that have not yet been addressed in the literature.

1.2 Project goals

The goal of this research is to implement and evaluate a privacy-preserving system using Fully Homomorphic Encryption (FHE) and Differential Privacy (DP). Specifically, our aim is to answer the following research questions:

- **Research Question 1:** How does combining FHE and DP affect the privacy guarantees of data subjects?
- **Research Question 2:** What potential vulnerabilities exist in a system combining FHE and DP, and under which conditions can these vulnerabilities enable re-identification of data subjects?
- **Research Question 3:** How does the performance of encrypted aggregation queries compare to non-encrypted aggregation queries?

- **Research Question 4:** How do different FHE schemes and libraries influence performance according to the evaluation criteria?

The main challenges are performance, accuracy, and privacy. Firstly, operations on Fully Homomorphically Encrypted data are known to be order of magnitudes slower than the equivalent operations on plain data. Secondly, there is an accuracy aspect, as DP adds random noise to datasets in order to protect individuals against membership inference attacks. Some FHE schemes, such as the CKKS scheme, are also approximate, meaning that homomorphic evaluations can amplify inaccuracies in the final output, which further increases the noise of the result [6]. Finally, there is a privacy aspect, which is the main reason of using both DP and FHE. The challenge is to preserve the privacy of the protected data, while striving for as high performance and accuracy as possible.

In this project, we set up an experimental testbed with three actors, a Healthcare center, a user who wishes to do data analysis, and an untrusted environment employed by the user for computation. This testbed aims to protect against these threats:

- **Eavesdropping attacks:** Traditional data analysis may encrypt data in transit and at rest, but process the data in an unencrypted way in memory, which opens up against eavesdropping attacks in untrusted environments. FHE prevents eavesdropping by also making sure eavesdropping is impossible on in-memory data processing.

- **Membership inference attacks:** Neither a legitimate user nor an adversary should be able to infer whether an individual's data was included in the dataset.

1.3 Limitations

There are three main limitations that should be noted in this project. First, although this work is regarding privacy-preserving computation on healthcare data, we use synthetic healthcare datasets to avoid ethical concerns associated with using real healthcare data. Additionally, no real healthcare data was available for this project.

Second, even though healthcare data usually includes textual data, such as names or demographic information, we focus on numeric data only. This is because homomorphic processing of textual data, although possible [7], represents a significant challenge that should be addressed in future work.

Third, the security analysis of this thesis focuses on attacks such as eavesdropping, chosen plaintext attacks, chosen ciphertext attacks, etc. Broader adversarial models which include insider attacks, denial-of-service attacks, and more, while important to consider in a real-life scenario, are out of scope for this work.

1.4 Disposition

This thesis is organized into five chapters. Chapter 1 introduces the project and its objectives. Chapter 2 provides the theoretical background necessary to place the results and discussion in context. Chapter 3 details the methods used in the study. Chapter 4 presents the experimental results and their analysis. Chapter 5 offers the conclusions drawn from the work.

2

Theory

This chapter presents the foundation for the privacy-preserving methods used in this thesis. We start by describing Fully Homomorphic Encryption (FHE) a cryptographic method to allow computations directly on encrypted ciphertext, and continue by describing Differential Privacy (DP), a statistical method to add noise to the output of an aggregate function over dataset, to protect the privacy of the dataset participants.

2.1 Fully Homomorphic Encryption

Before introducing fully homomorphic encryption, it is helpful to contrast it with public-key encryption schemes. In 1976, Diffie and Hellman introduced a definition for public-key encryption, which, unlike earlier encryption schemes, uses two keys, one for encryption (public key) and one for decryption (secret key) [8]. Specifically, public-key cryptography has three functions, $KeyGen$, which generates a public and secret key pair, Enc which encrypts a plaintext using the public key, and Dec which decrypts a ciphertext using the secret key [9]. This allows multiple actors to encrypt data using the public key, while it can only be decrypted by the holder of the secret key. Multiple public-key encryption schemes have emerged over the years, such as RSA in 1977 [9], and the ElGamal scheme in 1985 [10]. However, in public-key cryptography schemes, computations can not be performed directly on ciphertext. Practically, this means only the holder of the secret key (or an actor trusted by the holder) can perform computations on the data, after decrypting it.

Fully Homomorphic Encryption (FHE) is a method that allows computations, specifically multiplication and additions, to be performed directly on ciphertexts, producing encrypted results that decrypt to the same values obtained by operating on the underlying plaintexts [11]. This property allows the data to remain encrypted throughout the processing lifecycle, making FHE particularly valuable for privacy-preserving applications such as cloud computing, machine learning, and data analytics with sensitive data. It further enables data processors to compute on encrypted data without access to decryption keys or any knowledge of the underlying information.

In 1978, Rivest et al. proposed to create a cryptographic scheme that allows computations to be performed directly on the ciphertext, without intermediate decryption of the operands [12]. It was not until 2009 the first such scheme was created by

Gentry [13]. Since then, new FHE schemes have been created with significant performance improvements, and recent schemes include CKKS (Cheon-Kim-Kim-Song) scheme [6] and TFHE (Fast Fully Homomorphic Encryption over the Torus) [14].

The definition of Fully Homomorphic Encryption builds on the definition of public-key cryptography, but has an additional method, which evaluates ciphertexts *homomorphically*, meaning that computations performed on encrypted inputs produce a new ciphertext whose decryption matches the result of applying the same computation to the underlying plaintexts [11]. The definition of FHE is given in Definition 1, and is a condensed version of a definition given by Armknecht et al. in [15]. A visualization of the definition is given in Figure 2.1

Definition 1 (Fully Homomorphic Encryption) *A set of functions (KeyGen, Enc, Eval, Dec) with polynomial complexity that evaluate all binary circuits C with the definitions*

- $KeyGen(1^\lambda, \alpha) = (K_p, K_s, K_e)$*: Generation of public key* K_p*, secret key* K_s *and evaluation key* K_e *given the security parameter* λ *and auxiliary input* α*,*
- $Enc(K_p, m) = c$*: Encryption of to ciphertext* c *given* K_p *and plaintext message* m*,*
- $Eval(K_e, C, c_1, ..., c_n) = c$*: Evaluated ciphertext* c *given* K_e*, circuit* C*, and ciphertexts* $c_1, ...c_n$ *used as operands in* C*,*
- $Dec(K_s, c) = m$*: Decrypted message* m *given* K_s *and* c

such that the following is true

- $Dec(K_s, Enc(K_p, m)) = m$ *for all plaintexts* m*,*
- $Pr[Dec(K_s, Eval(K_e, C, c_1, ..., c_n)) = C(m_1, ..., m_n)] = 1 - error(\lambda)$*,*
- *there exists a polynomial* p*, such that for any key set* (K_p, K_s, K_e)*, any circuit* C*, and all ciphertexts* c*, the size of output of Eval is not more than* $poly(\lambda)$ *[15].*

The variable λ in Definition 1 is the security level, and is given in number of bits (e.g. 128-bit, 256-bit), which corresponds to the computational difficulty of brute force-attacking the secret key. Circuits are a set of connected logical binary gates (e.g. AND, XOR) connected in a way to calculate arbitrary operations, such as addition and multiplication [16].

In Definition 1, the correctness properties state that a plaintext that has been encrypted remains the same after decryption, but may alter by a small amount e after evaluations (called *error*), depending on the security parameter λ. The compactness property states that the size of the ciphertext only depends on λ, and the size does not change when evaluations are applied.

Compared to regular public-key encryption schemes, the FHE definition also defines an evaluation key for evaluating ciphertexts homomorphically. However, an alternative definition exists where there the public key is used for both decryption and homomorphic evaluation, so a separate evaluation key is not strictly required for an

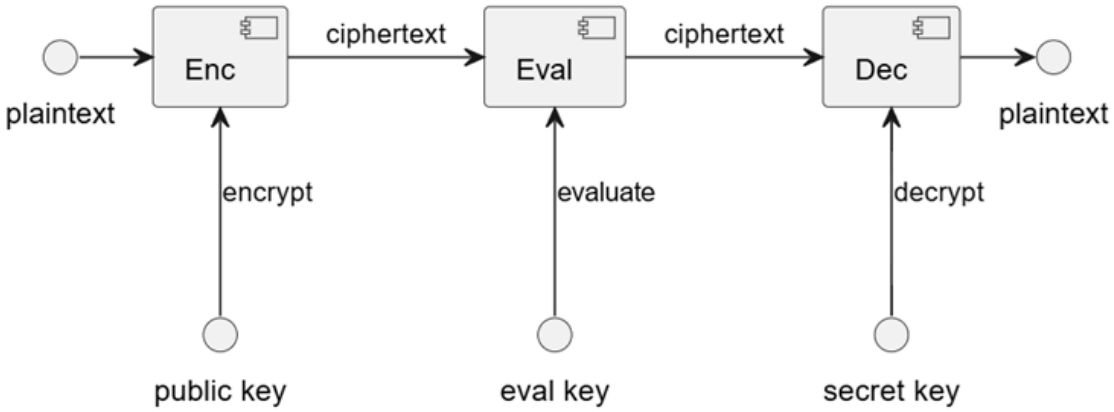

Figure 2.1: An example of Fully Homomorphic Encryption. The function *Enc* encrypts plaintexts with the public key. Ciphertexts can then be evaluated in *Eval* with the evaluation key, and decrypted in *Dec* with the secret key, resulting in their plaintexts. The evaluation key enables homomorphic evaluation by a third party which is neither the sender or receiver of the information, although in some definitions of FHE, the evaluation key is absent or part of the public key.

encryption scheme to be fully homomorphic [15]. A distinct evaluation key is however useful for scenarios where the actor that encrypts plaintext and the actor that performs homomorphic evaluations are different actors, and there is some kind of security requirement that is fulfilled by distinguishing the keys. Furthermore, there are schemes that even omit the evaluation key entirely, meaning that homomorphic evaluations can be done without a specific key [17].

2.1.1 Gentry's scheme

The first scheme that fulfills the definition of FHE given in Definition 1 was introduced by Craig Gentry in his 2009 PhD dissertation [18]. Thus, Gentry's work solved a long-standing problem in cryptography posed by Rivest et. al in 1978 [12].

Gentry's construction begins with a Somewhat Homomorphic Encryption (SHE) scheme, an encryption scheme that supports only a limited amount of additions and multiplications before the accumulated noise grows too large for correct decryption [18]. To address this limitation, Gentry introduced a *bootstrapping* procedure, which is an evaluation circuit that evaluates the schemes own decryption algorithm homomorphically. Recall from Definition 1 that each evaluation of ciphertext increases the noise, whereas decryption resets it. The key idea in Gentry's work is that decrypting the ciphertext homomorphically produces a ciphertext with less error, which enables an indefinite number of evaluations, as long as the bootstrapping operation is applied before the error grows too large [13]. With the addition of the bootstrapping operation, the Somewhat Homomorphic Encryption scheme becomes a Fully Homomorphic Encryption scheme. An algorithmic description of the bootstrapping operation is given in Algorithm 1.

However, for the bootstrapping operation to work, the bootstrapping operation it-

Algorithm 1
Bootstrapping. For setup, the secret key holder generates a decryption circuit and a *refresh key* K_r by encrypting the secret key. Then, the public key holder decrypts the ciphertext *homomorphically* using the decryption circuit, which resets the noise in the ciphertext. The refresh key is part of the public key in practice [19], but we separate the public and refresh key here for simplification.

Input: evaluation key K_e, decryption circuit C_{dec}, noisy ciphertext c, refresh key K_r
Output: refreshed ciphertext c'

function SetUp(K_s, K_p):
  $C_{dec} \leftarrow$ GenerateDecryptionCircuit()
  $K_r \leftarrow$ Enc(K_p, K_s)
  return $\{K_r, C_{dec}\}$

function Bootstrap(K_e, C_{dec}, c, K_r):
  $c' \leftarrow$ Eval(K_e, C_{dec}, c, K_r)
  return c'

self must not add noise to the extent that the ciphertext becomes impossible to decrypt. First, Gentry based his scheme on ideal lattices, a mathematical construction that supports multiplication and addition, and has a related encryption scheme with a shallow decryption circuit [18]. Second, Gentry used several methods to reduce the depth of the decryption circuit. Most notably, Gentry's scheme included a pre-processing in the encryption algorithm which offloads the computational overhead of the decryption algorithm to the encryption algorithm, thereby reducing the size of the decryption circuit to a point where the noise does not exceed the noise budget [18].

While Gentry's scheme demonstrated that FHE was really achievable, it was far from optimized in terms of performance, key sizes and ciphertext expansion. Depending on the chosen parameters, the bootstrapping operation could take several minutes to evaluate a single bit, and the public key alone could span several hundred megabytes [20]. As a result, Gentry's scheme was impractical for real-world usage. Nevertheless, Gentry's breakthrough paved way for future schemes to improve performance and reduce the size of keys and ciphertexts.

Aside from creating the first FHE scheme, Gentry also demonstrated that any Somewhat Homomorphic Encryption scheme (a scheme that can evaluate a circuit to a certain depth) is fully homomorphic if it can evaluate its own decryption circuit [18]. If a SHE scheme can evaluate its own decryption circuit, then a bootstrapping operation can be implemented for the scheme, which turns the bounded depth into an unbounded depth, as long as the bootstrapping operation is performed before the noise grows too large.

2.1.2 Hardness of FHE

Cryptographic schemes depend on hard problems, i.e. problems that are assumed to be computationally infeasible to solve. For example, the ElGamal encryption scheme depends on the hardness of solving discrete logarithms in cyclic groups [10]. Fully Homomorphic Encryption schemes generally depend on another type of problems, mainly Learning with Errors and variants of it.

2.1.2.1 Learning with Errors

FHE schemes generally depend on the hardness of a problem which is called Learning With Errors (LWE). The LWE problem was discovered by Regev in 2005 [21]. A definition of the problem is given in Definition 2.

Definition 2 (Learning With Errors) *Given access to a set of noisy linear equations of the form of*

$$\mathbf{As} \approx \mathbf{b} \pmod{q},$$

where matrix $\mathbf{A} \in \mathbb{Z}_q^{n \times m}$ *and vector* $\mathbf{b} \in \mathbb{Z}_q^n$ *are known to the adversary, the objective is to recover the secret vector* $\mathbf{s} \in \mathbb{Z}_q^n$*, where* q *is the coefficient modulus,* n *is the vector size and* m *is the lattice dimension [22].*

An example of Definition 2 is to recover the vector $\mathbf{s}$ from the following *approximate* random linear equations with $q = 21$, $n = 4$ and $m > 4$:

$$\begin{aligned}
18s_1 + 15s_2 + 4s_3 + 7s_4 &\approx 13 \pmod{21} \\
13s_1 + 19s_2 + 5s_3 + 2s_4 &\approx 11 \pmod{21} \\
5s_1 + 13s_2 + 4s_3 + 11s_4 &\approx 17 \pmod{21} \\
&\vdots \\
7s_1 + 2s_2 + 14s_3 + 6s_4 &\approx 4 \pmod{21},
\end{aligned}$$

The exact size of m, which determines how many equations there are in the equation system, seems to be insignificant for hardness [22]. Solving an equation system without noise is possible in polynomial time using the Gaussian elimination method [22]. However, the addition of noise to the equations in the LWE problem causes Gaussian elimination to unusable for solving the system due to the amount of noise accumulated after iterating over the entire system, particularly with high dimensions [22]. The noise property seemingly causes the LWE problem computationally infeasible to solve, making it suitable as the basis of cryptographic schemes.

To sample noise in the equation system in the LWE problem, Regev used the Gaussian distribution rounded to the closest integer with a standard deviation of $\sigma_e = q\alpha$, as shown in Figure 2.2, where $\alpha \in (0, 1)$ is the tuning parameter for the noise distribution, typically inversely proportional to the size of the secret key [22]. The standard deviation should therefore be tuned according to both the chosen modulus and size parameters, with a higher q increasing the standard deviation and a higher n decreasing the standard deviation.

The LWE problem is defined by several parameters which influence the security and performance of cryptographic schemes based on the problem. These parameters are listed in Table 2.1.

Symbol	Parameter description
n	Secret vector size
q	LWE modulus
α	Noise coefficient for lattice
m	Lattice dimension
σ_s	Secret key sample distribution, typically $\sigma_s = q\alpha$
σ_e	Noise standard deviation, typically $\sigma_e = q\alpha$

Table 2.1: Parameters of the LWE problem. Generally, n, q and α are the properties that signify an instance of LWE [23], whereas the rest of the parameters are derivative or have no significance. For example, the size of m seems to be insignificant for hardness [22], whereas the standard deviations σ_e and σ_s are dependent on the value of q and α.

Regev used LWE as the basis to create an asymmetric (but non-homomorphic) encryption scheme [22]. Furthermore, Regev states that in order to guarantee correctness and security, q should be a prime between n^2 and $2n^2$, lattice dimension should be $m = 1.1n \log q$ and noise coefficient should be $\alpha = 1/(\sqrt{n} \log^2 n)$ [22].

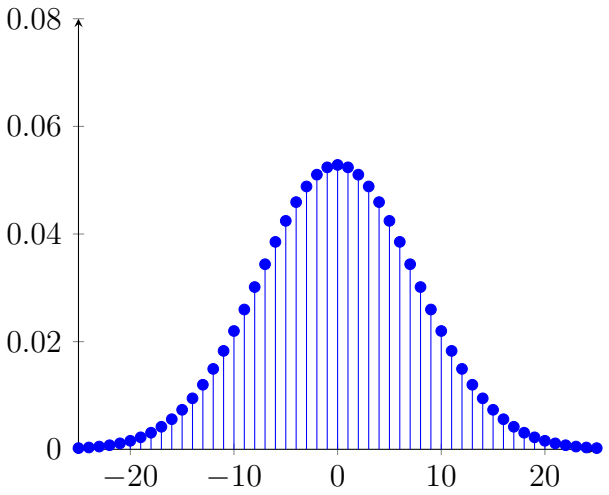

Figure 2.2: In Regev's original formulation of the LWE problem, the noise for the equations in the Learning With Errors problem was sampled from a discrete Gaussian distribution (continuous Gaussian distribution rounded to the nearest integer). In the distribution shown in the figure, the standard deviation is $\sigma = q\alpha$, where $q = 151$ and $\alpha = 0.05$. Later works question the security of using this distribution due to the presence of timing attacks [24].

The LWE problem is believed to be difficult to solve for different reasons; all known algorithms that solve the problem run in exponential time, and is believed to be as hard as the worst-case scenario of the shortest vector problem [22]. However, estimating the actual hardness of the LWE problem under different chosen parameters

is difficult, as there is a lack of closed formulas that accurately estimate the time it takes to break the scheme [25].

2.1.2.2 Learning with Errors over Rings

The main drawback of schemes based on the Learning With Errors (LWE) problem that they tend to be inefficient, with key sizes and computation times that grow at least quadratically based on the chosen security parameter [26]. To address this limitation, Lyubashevsky et al. created a ring variant of LWE known as Learning With Errors over Rings (RLWE), which possesses similar security guarantees, but leads to smaller keys and faster computation times in schemes using this problem [26].

The core idea behind RLWE is that, unlike LWE, the secret is a polynomial rather than a vector, and the secret polynomial is sampled from a quotient ring. The RLWE problem is based on a category of quotient rings with the structure $\mathbb{Z}_q[x]/(x^n+1)$, which has a unique representative for each polynomial modulo q of degree less than n. That is, polynomials where each term has a power below n and a coefficient below q. For example, let $n = 3$ and $q = 2$, then the quotient ring is the following set of polynomials:

$$\mathbb{Z}_2[x]/(x^3+1) = \{0, 1, x, x+1, x^2, x^2+1, x^2+x, x^2+x+1\}$$

By tuning the parameters n and q, different quotient rings can be created, which is used in the RLWE problem. The problem is defined in Definition 3.

Definition 3 (Learning With Errors over Rings) *Let the quotient ring $R_q = \mathbb{Z}_q[x]/(x^n+1)$, and n be a power of 2. Pick m samples of $a_i(x) \in R_q$ and $b_i(x) \approx a_i(x) * s(x)$. Given the set of m samples in the form of*

$$\begin{gathered}(a_1(x), b_1(x)) \\ (a_2(x), b_2(x)) \\ \vdots \\ (a_m(x), b_m(x)),\end{gathered}$$

the goal is for the adversary to recover the secret polynomial $s(x)$ [27].

2.1.2.3 Noise distributions in LWE and RLWE

Both the LWE and RLWE problems rely on sampling noise into an equation system to create a hard problem. Without noise, the problems cease to be computationally difficult to solve. Regev sampled the noise for the LWE problem from discrete Gaussian distributions [21], as discussed in Section 2.1.2.1. However, the sampling operation for Gaussian distributions is expensive and susceptible to timing attacks [28], making Regev's asymmetric encryption scheme vulnerable against side-channel attacks.

In 2013, Cabarcas et al. introduced the first provably secure LWE-based asymmetric scheme [24]. The main contribution of the scheme was sampling the noise from a narrow and uniform distribution, instead of a Gaussian distribution [24], which eliminates timing attacks and is computationally efficient.

In 2018, Bos et al. introduced Kyber, a key-encapsulation mechanism based on a variant of LWE known as Module-LWE. In Kyber, the secret and noise are sampled from a binomial distribution [29], which is faster than discrete Gaussian sampling while still preserving required security properties.

2.1.3 Evolution of FHE schemes

While Gentry's scheme solved the problem of error growth with the bootstrapping method, the bootstrapping method itself was slow enough to make its usage untenable in practice. Following Gentry's breakthrough, several schemes would be proposed over the years, which improves different aspects of the scheme proposed by Gentry. Schemes can generally be sorted into two different categories, leveled schemes and bootstrapping schemes, defined in Definition 4 and Definition 5, respectively, although a combination of both strategies is also possible [30].

Definition 4 (Leveled scheme) *FHE scheme that supports correct evaluation of all circuits up to a specified depth L, determined during key generation.*

Definition 5 (Bootstrapping scheme) *FHE scheme with a bootstrapping procedure for refreshing accumulated errors in ciphertexts, enabling evaluation of circuits of unbounded depth.*

A leveled scheme that is not a bootstrapping scheme can only evaluate ciphertexts up to a certain depth, after which the error exceeds the error budget. Bootstrapping schemes can evaluate circuits of any depth, provided that the bootstrapping operation is performed each time the error is about to exceed the error budget. Therefore, the depth of the circuit must in leveled schemes be decided beforehand. There are two main benefits of still using the bootstrapping method in leveled schemes. Firstly, it enables flexibility by supporting circuits of unbounded depth. Secondly, depending on the scheme, it may be cheaper to perform a bootstrapping operation than evaluating a deep circuit.

There is no FHE scheme that takes advantage of all breakthroughs discovered since the introduction of Gentry's scheme, and each scheme has different strengths and weaknesses [31]. As a result, schemes differ in key size, ciphertext expansion, evaluation efficiency, noise growth, and other performance characteristics.

FHE schemes can be broken into four generations [31], where Gentry's scheme is part of the first Generation. The second generation of FHE schemes includes the BV (Brakerski-Vaikuntanathan) [32], BFV (Brakerski/Fan-Vercauteren) [33], and BGV (Brakerski-Gentry-Vaikuntanathan) [19] schemes, among other schemes. The third generation of schemes consists of GSW (Gentry-Sahai-Waters) [34], FHEW [35] and TFHE (Fast Fully Homomorphic Encryption over the Torus) schemes, and the fourth and the latest generation includes the CKKS scheme [6]. In the following

subsections, a few schemes from each generation, are described in further detail.

2.1.3.1 Early FHE schemes

Recall from Section 2.1.1 that Gentry managed to make the decryption circuit shallow enough by "squashing the circuit", i.e. letting the encryptor do part of the decryption work through a special pre-processing step. This method reduced the depth of the homomorphic decryption circuit to the extent that the scheme became bootstrappable, and therefore fully homomorphic. The security of the circuit-squashing method relied on the assumption of the hardness of the Sparse Subset Sum Problem (SSSP). However, Lee showed that the specific form and parameters used in Gentry's scheme made the problem solvable within a few days, although the problem could be remedied with better parameters [36]. Nevertheless, relying on multiple hardness assumptions can weaken an encryption scheme and complicate its security analysis, especially for hardness assumptions that are not yet well understood.

In 2011, Brakerski and Vaikuntanathan created the BV (Brakerski-Vaikuntanathan) scheme, which uses a *dimension-modulus reduction* method to achieve a bootstrappable scheme, instead of the circuit squashing method used by Gentry [32]. With the dimension-modulus reduction method instead of squashing the circuit, Brakerski and Vaikuntanathan avoids the Sparse Subset Sum Problem altogether, and bases the hardness assumption solely on the LWE problem [32], described in Section 2.1.2.1.

The dimension-modulus reduction method works like the following: starting from a ciphertext with dimension n and modulus q (i.e. in the form $\mathbb{Z}_q^n \times \mathbb{Z}_q$), the dimension-modulus reduction method is to transform the ciphertext to the approximately same expression in with smaller modulus $p < q$ and lower dimension $k < n$. Decrypting this simpler ciphertext brings down the depth of the decryption circuit to the extent that the scheme becomes bootstrappable, though at the cost of a loss in precision due to the compression introduced by the technique. It turns out that the modulus reduction method on its own (without dimension reduction) is a very useful method for decreasing the noise in a ciphertext, though at the expense of accuracy. The modulus reduction method has therefore been used in many other schemes, and a definition is given in Definition 6.

Definition 6 (Modulus switching) *For a chosen modulus q, select a scaling factor γ and convert an LWE-based ciphertext $\mathbf{c}$ from $\mathbb{Z}_q$ to a lower modulus $\mathbb{Z}_{q/\gamma}$ by scaling each term of the ciphertext by a factor of $\lfloor \mathbf{c}/\gamma \rceil$ [37].*

Another feature of the BV scheme is the *re-linearization* method [32]. Recall that the LWE problem is basically a system of linear equations. Now, multiplying two linear equations (which corresponds to a homomorphic multiplication) creates a quadratic expression, which causes significant ciphertext expansion, since the ciphertext now needs to hold the new coefficients in the quadratic expression. The BV scheme addresses this problem with re-linearization, which reduces the ciphertext back to (nearly) its original size $n+1$. Re-linearization works by transforming the quadratic expression back to a linear one, and using more than one secret key based on the depth of the evaluation circuit. For a circuit of depth L, L secret keys can be used.

The authors of the BV scheme also created a variant of Gentry's scheme in 2011 which retains the squashing method [38], but based on the RLWE problem. However, as explained previously, this again introduces the hardness assumption of the sparse subset sum problem.

2.1.3.2 BFV: The Brakerski/Fan-Vercauteren scheme

Brakerski and Vaikuntanathan created two schemes, one that introduces the dimension-modulus reduction technique and one derived from Gentry's scheme that is based on the RLWE problem. In 2012, Fan et al. introduced the BFV scheme (Brakerski/Fan-Vercauteren), which combines the dimension-modulus reduction and re-linearization technique with the RLWE problem, implemented by the two schemes separately [33].

Additionally, the BFV scheme introduces optimizations on the re-linearization technique. In fact, it introduces two different re-linearization techniques [33]. These optimized techniques reduce the re-linearization key sizes and faster computation [33].

2.1.3.3 BGV: The Brakerski-Gentry-Vaikuntanathan scheme

In 2014, Brakerski, Gentry, and Vaikuntanathan introduced the BGV scheme, which is a leveled scheme that avoids bootstrapping altogether [19]. The downside is that the depth L needs to be specified beforehand.

Similar to the BV scheme described in Section 2.1.3.1, the BGV scheme incorporates a modulus switching technique. However, in this scheme, the modulus reduction happens iteratively at each level of the evaluation circuit, compared to the BV scheme where the entire reduction happens just before the bootstrapping operation [19]. The purpose of modulus switching is to reduce the noise, and the core idea behind the BGV scheme is to decide the circuit depth L beforehand and then picking the moduli to be used at each level of the circuit, such that the entire circuit can be evaluated before the noise grows too large [19].

However, as mentioned previously, bootstrapping may be used even in leveled schemes. In the BGV scheme, the execution time increases exponentially depending on the chosen depth, meaning that computations in deep circuits become expensive [19]. The bootstrapping operation can therefore be reintroduced as an optimization to eliminate the need to specify the depth beforehand, increase performance of deep circuits, and reduce the length of ciphertexts.

The BGV scheme also uses a technique called *batching*. The core idea behind this tehcnique is that a single ciphertext can hold multiple plaintexts, and therefore can be evaluated simultaneously [19], enabling SIMD (Single Input Multiple Data) operations. Interestingly, batching can be applied to the bootstrapping operation, which for the BGV scheme speeds up the bootstrapping operations significantly [19].

2.1.3.4 TFHE: Fast FHE over the torus

Before describing the TFHE (Fast Fully Homomorphic Encryption over the Torus) scheme, we start with briefly describing a scheme which has an important part used

in the TFHE scheme. This scheme is the GSW (Gentry-Sahai-Waters) scheme [34].

The GSW scheme is a leveled scheme and introduces two novelties. Firstly, this scheme does not require an evaluation key to evaluate circuits homomorphically. Secondly, it introduces the *approximate eigenvector method*. This scheme encodes a plaintext m over a matrix C of size $n \times n$ over $\mathbb{Z}_q$ for a chosen dimension n and modulus q. The secret is a vector $\mathbf{s} \in \mathbb{Z}_q^n$, and the plaintext m is hidden in the construction $C \cdot \mathbf{s} \approx m \cdot \mathbf{s}$, which makes the secret $\mathbf{s}$ an approximate eigenvector [34]. This construction supports homomorphic addition and multiplication by simple matrix addition and matrix multiplication. The structure of the ciphertext is important for understanding the TFHE scheme.

In 2016, the TFHE scheme was created by Chillotti et al. and is based on the GSW scheme [14]. The TFHE scheme is a bootstrapped scheme which is based on a RLWE variant of the GSW scheme. Interestingly, the TFHE scheme expresses a ciphertext as a product of a LWE and a GSW ciphertext, and uses this to derive a bootstrapping method that is faster than previous generations [14]. Aside from the bootstrapping operation, the TFHE scheme contains optimizations in ciphertext expansion and homomorphic evaluation of lookup tables [14].

2.1.3.5 CKKS: The Cheon-Kim-Kim-Song scheme

In the fourth and most recent generation is the CKKS scheme (Cheon-Kim-Kim-Song), a leveled scheme that encrypts real (floating point) numbers numbers rather than integers [6]. A distinguishing feature of the CKKS scheme is that it is an approximate FHE scheme, meaning that decrypting a ciphertext returns the *approximate* plaintext [6]. The support the CKKS provides for floating point numbers makes it useful for privacy-preserving machine-learning use cases [39].

The CKKS scheme introduces two new methods, namely rescaling and RLWE-based batching. Rescaling addresses the problems of exponential ciphertext expansion with circuit depth by reducing the ciphertext modulus, which brings down the ciphertext expansion linearly with the circuit depth [6].

In its original formulation, the CKKS scheme did not contain a bootstrapping method, as the decryption circuit is too deep to evaluate homomorphically. Subsequent research achieved a bootstrapping operation by approximating the modulus switching method using a *scaled sine function* [40]. Compared to previous schemes, CKKS achieves the fastest bootstrapping operation [41]

A caveat to the CKKS scheme is that there exists a chosen plaintext attack which can be used to recover the key if there is a decryption oracle present [42]. However, this attack is eliminated by adding noise to the output of the decryption operation [43], though at the expense of additional inaccuracy.

2.1.4 FHE libraries

There exists multiple FHE libraries, including OpenFHE [44], SEAL [45], Concrete [46], and HELib [47]. These libraries vary in what FHE schemes they support,

the implementation language, the development interface, and the level of documentation and support.

OpenFHE is a successor of a previous library called PALISADE. It implements BFV, BGV, CKKS, and TFHE schemes. It is written in C++ as an open-source library [44]. SEAL stands for *Simple Encrypted Arithmetic Library* and is an open-source library developed in C++ by Microsoft Research which implements BGV, BFV, and CKKS schemes [45]. Concrete is a open-source library written in Python and Rust, which implements the TFHE scheme [46]. HELib is an open-source library written in C++, and implements CKKS and BGV schemes.

In this project, we use the Zama Concrete library for evaluating the TFHE scheme and a Python wrapper for SEAL called SEAL-Python for evaluating the BFV, BGV and CKKS schemes [48].

2.1.4.1 Concrete: the TFHE library

The Concrete library is an FHE library for Python that implements the TFHE scheme. An interesting feature of this library is that circuits are compiled code written as if it were in the unencrypted domain, where addition and multiplication operators are overloaded to perform homomorphic addition and multiplication. This results in easily readable Python code, but there are limitations on operands and supported operations within the homomorphic domain.

Concrete supports the usual linear operations like addition and multiplication, and also implements support for non-linear operations such as comparisons by using Look-Up Tables (LUTs). Some operations can compile LUTs out of the box, but an interface is provided for developers to create custom LUTs for unsupported non-linear functions. However, there are many limitations in Concrete which makes it difficult to implement queries for data analysis, particularly with large amount of data. Some examples of the limitations include:

- **Supported operations:** There is a limitation on the operations that are supported by Concrete on encrypted operands. For example, dividing an encrypted variable with another encrypted variable is not supported.
- **Floating-point arithmetic:** Floating point arithmetic is supported as an intermediary step in Concrete circuits, but the input and output of circuits can only be integers (as the underlying TFHE scheme is an integer encryption scheme).
- **Bit-width:** There are several limitations on the bit-width of the operands supported by the Concrete library. Non-linear functions are implemented with LUTs, but LUTs only support 16-bit operands. Therefore, the maximum bit-width of circuits containing non-linear operations is 16 bits.
- **Comparisons:** Concrete supports compiling circuits with comparative operations (such as Less-Than and Greater-Than). However, Concrete does not support branching, and instead the developer must build in the comparative functions in the circuit arithmetic (like multiplication or addition with 1 or 0

depending on the condition outcome). Comparative operations are also very expensive, and Concrete supports different strategies for implementing the comparisons efficiently, which depends on how the variables in the comparison is used in the rest of the function.

- **Branching and loops:** For loops are supported, but not if-else statements or while loops, and for loops can also not be exited early. On the other hand, this ensures that the circuit takes the same time to execute regardless of input variables, which mitigates timing attacks.

Implementing a query in Concrete typically runs into one or several of these limitations, and tackling all limitations simultaneously requires techniques such as bit-shifting, normalization, algorithmic approximation methods (such as Newton's method), etc.

2.1.4.2 SEAL: Microsoft's FHE Library

Microsoft's SEAL library implements support for the BGV, BFV, and CKKS schemes. SEAL uses a different approach than for implementing homomorphic evaluation circuits. In the SEAL library, developers construct circuits with a chain of piecewise circuits which implement homomorphic addition, subtraction, multiplication, and other operations.

The downside is that it is up to the user to manage noise growth and circuit complexity. For example, SEAL provides support for re-linearization to minimize the complexity of deep circuits. It is however up to the user to implement re-linearization in the evaluation circuit. Since re-linearization is an expensive operation, the re-linearization strategy must be carefully chosen to minimize the overhead.

2.2 Differential privacy

Differential Privacy (DP) is a statistical method that ensures that the output of a computation over data does not change significantly when a single data subject is added or removed, thereby protecting individual privacy, while still preserving the accuracy of aggregations [49]. Therefore, DP protects against membership inference attacks [50].

Differential Privacy is achieved by introducing noise into the result of a query function, which protects individual data points by making it statistically difficult to determine the value of individual data points [51]. The noise which should be added to the output is generated by picking randomly from a specific statistical distribution [52].

Research of Differential Privacy started as an attempt in 2006 by Dwork et al. to add privacy guarantees to databases [52]. This is called ϵ-Differential Privacy, with ϵ being the leakage or privacy budget [52]. A smaller ϵ gives stronger privacy guarantees, while a larger ϵ allows more accurate query results. A definition of ϵ-Differential Privacy is given in Definition 7.

Definition 7 (ϵ-Differential Privacy) *Let D_1 and D_2 be two datasets that differ by one record. A randomized function κ is said to provide ϵ-Differential Privacy (ϵ-DP) if*

$$e^{-\epsilon} \leq \frac{Pr[\kappa(D_1) = R]}{Pr[\kappa(D_2) = R]} \leq e^{\epsilon}$$

where ϵ is the privacy budget and R is a single sample of the randomized function κ [53].

There are different mechanisms which implement Differential Privacy, such as the Laplace mechanism and the Gaussian mechanism. However, they differ in the type of definition they fulfill. The Laplace mechanism satisfies Definition 7 [54]. The Gaussian mechanism satisfies (ϵ, δ)-DP, a relaxed version of ϵ-DP that allows a small probability δ of exceeding the privacy budget [55]. The Gaussian mechanism does not guarantee privacy, but is easier to understand and analyze, and does not actually experience the loss of privacy guarantee in practice with a small enough δ chosen [55].

2.2.1 Laplace mechanism

The Laplace mechanism implements ϵ-Differential Privacy, which guarantees the privacy of data set participants [51]. The Laplace mechanism relies on the Laplace distribution, defined in Definition 8.

Definition 8 (Laplace distribution) *The Laplace distribution is defined as follows [56]:*

$$Lap(x \mid b) = \frac{e^{-|x|/b}}{2b},\ b > 0.$$

The parameter b affects the amplitude of the distribution. An example of the Laplace distribution is shown in Figure 2.3. It is possible to generate noise which is ϵ-differentially private by drawing a sample from a Laplace distribution with appropriately set parameters.

The amount of additive noise added by the Laplace mechanism depends on two factors: the privacy budget, and the L_1 *sensitivity* of the aggregate computation done on the query function, which is based on the L_1 norm. The L_1 norm and L_1 sensitivity are defined in Definition 9 and 10, respectively.

Definition 9 (L_1 norm) *The L_1 norm of a vector is defined as the sum of the absolute value of its components:*

$$||\mathbf{x}||_1 = \Sigma_i |x_i|$$

where $||\cdot||_1$ denotes the L_1 norm [57].

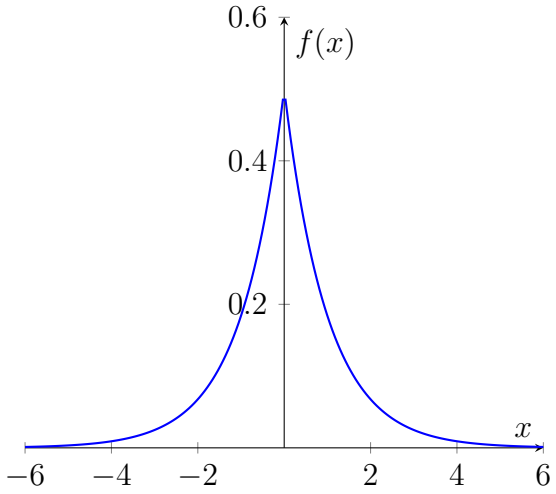

Figure 2.3: Graph plotting the Laplace distribution with $b = 1$. The noise in ϵ-Differential Privacy can be sampled from Laplace distributions.

Definition 10 (L_1 sensitivity) *The L_1 sensitivity of a function f, denoted Δf, is based on the largest change a single record can achieve to the output of the function, and is defined as follows [54]:*

$$\Delta f = \max_{D_1, D_2} ||f(D_1) - f(D_2)||_1,$$

where D_1 and D_2 are datasets that differ by one record [54].

Using Definition 8 and Definition 10, we can now define the Laplace mechanism in Definition 11 as a function that adds noise to the output of a query function depending on the privacy budget and the L_1 sensitivity of said function.

Definition 11 (Laplace mechanism) *The Laplace mechanism is a function that adds noise to the output of query function $f(x)$:*

$$M_L(f(\cdot), \mathbf{s}, \epsilon) = f(\mathbf{s}) + R,$$

where R is a sample picked from the distribution $Lap(x|\frac{\Delta f}{\epsilon})$ [55].

The L_1 sensitivity of different query functions changes the Laplace distribution significantly. To illustrate the significance, two common query functions can be compared:

- $\Delta count = 1$, since each record can add at most one to the output.
- $\Delta sum = max(\chi)$, where χ is the set of possible values in the dataset, since each record can at most add the maximum value of all possible values.

The variance of Laplace distributions is $\sigma^2 = 2b^2$ [56]. The variance of the Laplace distribution in the Laplace mechanism is therefore $\sigma^2 = 2(\frac{\Delta f}{\epsilon})^2$. Assuming that $max(\chi) = 100$, $M_L(count(\cdot), \mathbf{s}, 1)$ samples from a distribution with a variance of $\sigma^2 = 4$ and $M_L(max(\cdot), \mathbf{s}, 1)$ samples from a distribution with a variance of $\sigma^2 = 20000$. Therefore, it is important to apply the correct sensitivity to the Laplace

mechanism, and each query function has its own sensitivity, that must be calculated properly.

2.2.2 Privacy budget

The amount of noise that should be added to the output of a query is defined by the privacy budget ϵ, where a lower budget gives stronger privacy but noisier query result. A question naturally arises; what prevents an adversary from submitting the same query over and over again, and estimating the mean from the samples? A central problem in Differential Privacy is the fact that multiple queries over the same dataset continuously degrade privacy guarantees, until an adversary can infer the original data [58] (known as *sequential composition*).

Differential Privacy addresses sequential compositions by lowering the privacy budget each time the user sends a query, so each query samples from a different distribution, and also becomes less accurate. This prevents any users from gaining knowledge about individual data points from multiple related queries. One such implementation is to define a privacy cost for each query and subtracting the cost from the budget after each execution, and reject queries whose costs exceed the privacy budget [59].

Another important aspect is the initial value of the privacy budget. The lower the privacy budget is, the more noise is added, but the exact amount of noise that should be added to the result of a query is mainly a social question, and it is hard to define a threshold for the amount of noise that should be added to make the privacy "good enough" [54]. On one hand, analysts desire a high privacy budget to produce accurate results, but on the other hand, accurate results could lead to individuals' privacy being breached. Even worse, there is no method that directly measures the privacy gained by the privacy budget in a meaningful sense in practice [60]. Therein lies the difficulty for a data owner to set an appropriate privacy budget given the sensitivity of the data, the data domain, and risks associated with exposing the data [60].

3

Methods

To answer the Research Questions of the project, this project simulates a real-world scenario involving data analysis in the healthcare domain. The simulation is set up as follows: Policymakers often require access to sensitive medical data to make informed decisions on insurance or healthcare policy. However, medical data are sensitive and insurance providers may not be able to share the data with third parties. Furthermore, insurance providers may not be able to generate the statistics themselves due to a lack of resources or other practical reasons. This creates a need for privacy-preserving solutions that allow policy makers to analyze data without knowing anything about the underlying data subjects. This chapter presents experimental design, the dataset, the parameter configurations, and the evaluation criteria.

3.1 Scenarios

The experiments in this project considers four different scenarios with different combinations of Fully Homomorphic Encryption and Differential Privacy:

- Scenario S_1: Baseline, no homomorphic encryption nor differential privacy processing
- Scenario S_2: Differential privatization applied to data analysis
- Scenario S_3: Fully Homomorphic Encryption applied to data set
- Scenario S_4: Fully Homomorphic Encryption applied to dataset, and Differential Privatization applied to data analysis

These four scenarios were selected to isolate and compare the combined effects of Fully Homomorphic Encryption and Differential Privacy on performance, security, and accuracy. Scenario S_1 serves as a baseline, providing a setting against which all other configurations can be measured. Scenario S_2 allows us to evaluate the impact of applying DP alone, whereas Scenario S_3 allows us to evaluate the impact of applying FHE alone. Finally, Scenario S_4 examines the full pipeline in which encrypted data is protected end-to-end and privacy noise is applied during analysis.

3.2 Dataset and processing

The experiments use a synthetic dataset D designed to simulate sensitive personal and financial information. The attributes included in D consist of numeric fields such as income, health care coverage, and health care expenses, all measured in USD. These values allow us to model analytical tasks related to healthcare insurance. The dataset is generated with Synthea, a simulator for generating synthetic healthcare data for experimental purposes [61].

The dataset contains three columns: healthcare expenses, healthcare coverage, and income, which are represented with exp_i, cov_i, and inc_i, respectively, where i is the ith record of D. Samples from this dataset is shown in Table 3.1.

#	**Healthcare expenses ($)**	**Healthcare coverage ($)**	**Income ($)**
1	57934.77	0.00	123160
2	1812.82	21266.58	1371
3	3291411.70	503097.69	156004
4	512838.40	287825.94	102324
5	14338.47	680.31	41452
⋮			

Table 3.1: Samples from the dataset D used in this project. This dataset contains numerical data, but no textual data due to limitations in homomorphic processing of textual data.

However, processing textual data homomorphically, such as names, addresses or demographic information, is out of scope in these experiments. The reason for this is that textual processing typically used for conditional operations, such as filtering, which have poor support in common FHE libraries. This is a rather big limitation, since filtering on demographic information is a key aspect in shaping healthcare insurance policies. Nevertheless, not including demographic data in the experimental dataset also has a security benefit, namely that it makes membership inference attacks more likely.

3.2.1 Dataset size

The experimental dataset comes in three different sizes; large, medium, and small. The small dataset contains 10^2 records, the medium dataset contains 10^3 records, and the large dataset contains 10^4 records. The purpose of using different sizes is to explore how the evaluated metrics change when as the order of magnitude increases, as scaling is an important factor for analysis of big amounts of data.

3.2.2 Data encryption

In the scenarios with Fully Homomorphic Encryption, each attribute in D is encrypted homomorphically with the same keys, and swapped in place of the unencrypted value. These operations yield the encrypted dataset $Enc(D)$. Variants of

the encrypted dataset is created using different schemes and encryption parameters. Four different commonly used FHE schemes are evaluated in this thesis. These are BGV, BFV, TFHE and CKKS, which represent schemes across different generations, using both the bootstrapping and leveled approaches. Datasets that are encrypted with these schemes are denoted $Enc_{bgv}(D)$, $Enc_{bfv}(D)$, $Enc_{tfhe}(D)$, and $Enc_{ckks}(D)$, respectively. For brevity, we omit cryptographic keys from the notation in this chapter, although they are implicitly part of all encryption, evaluation, and decryption operations. Each of these encryption operations are fine tuned with appropriate encryption parameters, which differ based on the scheme that is used. As a baseline to compare the different encrypted variants of the dataset with, the unencrypted dataset D is used.

For the TFHE scheme, the Concrete library is used for implementing evaluation of the queries, whereas for the BGV, BFV and CKKS schemes, Microsoft SEAL is used. Concrete is used for TFHE as it is the main library for TFHE implementation, whereas SEAL is used for the other schemes as it provides general support for many schemes. These libraries also use different interfaces, which enables comparison on developer experience. Finally, both libraries enjoy active development support.

3.2.3 Differential privacy processing

In the experiments with Differential Privacy, the query output undergoes a DP processing step by adding noise to the result after homomorphic evaluation of the query and decryption of the query result. The library is used for this step is IBM's differential privacy Python library *diffprivlib* [62].

The noise mechanism used in this project is the Laplace mechanism M_L, as described in Section 2.2. The amount of noise that should be added depends on the sensitivity of the aggregate function f, denoted Δf, and the privacy budget parameter ϵ. There is no general method for determining the appropriate value of the ϵ. The problem of setting ϵ is described as a "social question", i.e. what is generally considered a good balance between privacy and accuracy [52]. Consequently, the selection of ϵ depends on the relative importance placed on preserving privacy in a given application. To guide our choice, we examined real-world scenarios where DP has been applied to protect datasets containing demographic information.

One such scenario is the COVID-19 mobility report dataset by Google, who set $\epsilon = 0.11$ on a dataset containing detailed demographic information [63]. The 2020 US census uses a very different value of $\epsilon = 19.61$ [64]. A lower value of ϵ leads to stronger privacy, but weaker accuracy, and vice versa for a higher value of ϵ.

3.2.4 Aggregate queries

The processed datasets described in Section 3.2 is subjected to a set of aggregate queries. These queries are the following:

- $f_1(D)$: What is the sum of expenses across all patients?
- $f_2(D)$: What is the average income after healthcare expenses for patients?

- $f_3(D)$: What is the average coverage of healthcare expenses?

These queries were chosen because they realize different homomorphic computations on the data and are commonly used in data analysis. They use a variety of operations, such as addition, division, and multiplication, which can be evaluated homomorphically.

3.2.5 Requirements

The requirements for the system are stated in Table 3.2. These requirements were selected to capture the availability and confidentiality concerns. Requirement 1 and 2 capture the availability requirements to make data usable and valuable, whereas Requirement 3 and 4 capture the confidentiality requirements to protect privacy of individual patients.

Label	**Requirement**
REQ1	Trusted external users shall be provided with patient data for analysis.
REQ2	Aggregated data shall be available in a decrypted format.
REQ3	Patient data shall be encrypted outside the trusted domain.
REQ4	Aggregated data shall be protected from membership inference attacks.

Table 3.2: Functional requirements for the system.

3.2.6 Process and actors

In this model, there are three actors: the Healthcare center, the user and the untrusted cloud. The healthcare center is obliged to keep the personal information of its patients private, but would still like to share aggregate data to users that have a valid use case for the data. Therefore, the healthcare center employs privacy-preserving methods for protecting the data, to balance the counteracting interests.

The user is honest-but-curious, meaning that the user does have a valid reason to process the aggregate data, and does not interfere with the process. However, the user is also interested in identifying the private information of the users who are included in the dataset. To this end, the user can decrypt any ciphertexts using the decryption interface provided by the healthcare center.

The untrusted cloud is a cloud service employed by the user. This environment is compromised by a malicious agent, who wishes to reveal as much information about the data as possible. This agent has full insight into the process memory, database, and network traffic. The agent does not have access to the decryption interface provided by the Healthcare center. The agent is not permitted by the healthcare center to view aggregate-level data nor data of individuals.

For Scenario S_1 and S_2, the process is the following:

1. The healthcare center publishes the raw dataset into a database
2. The user queries the database and gets the result
3. In S_2, the result in Step 2 is perturbed with Differential Privacy

For Scenario S_3 and S_4, the process is the following:

1. The healthcare center encrypts the raw data to publish a homomorphically encrypted dataset.
2. The user retrieves the dataset and the evaluation key from the healthcare center.
3. The user copies the dataset and the evaluation key to the untrusted cloud server so it is ready for computations.
4. The user submits a query to the untrusted cloud server.
5. The untrusted cloud server computes the user-submitted query.
6. The untrusted cloud server returns the result to the user.
7. The user request decryption of the encrypted result to the decryption interface provided by the Healthcare center over an encrypted channel.
8. The healthcare center authenticates the user and decrypts the encrypted result. Additionally, the healthcare center adds noise with the Laplace mechanism to the result in Scenario S_4 only.
9. The healthcare center returns the result of the aggregation queries to the user over an encrypted channel.

For Scenario S_4, the process of how the user interacts with the healthcare center and the untrusted cloud is shown in Figure 3.1, in the order the operations are done.

3.2.7 Experimental summary

In summary, the scenarios are represented with the following equations, where $f(\cdot)$ is an arbitrary aggregate query on the dataset and M_L is the Laplace mechanism with sensitivity Δf and privacy budget ϵ, and Enc, Dec and $Eval$ are functions belonging to homomorphic encryption schemes:

- Scenario S_1: $R = f(D)$
- Scenario S_2: $R = M_L(\Delta f, \epsilon, f(D))$
- Scenario S_3: $R = Dec(Eval(f, Enc(D)))$
- Scenario S_4: $R = M_L(\Delta f, \epsilon, Dec(Eval(f, Enc(D))))$

Note that different parts of the equations are processed by different actors, described in Section 3.2.6.

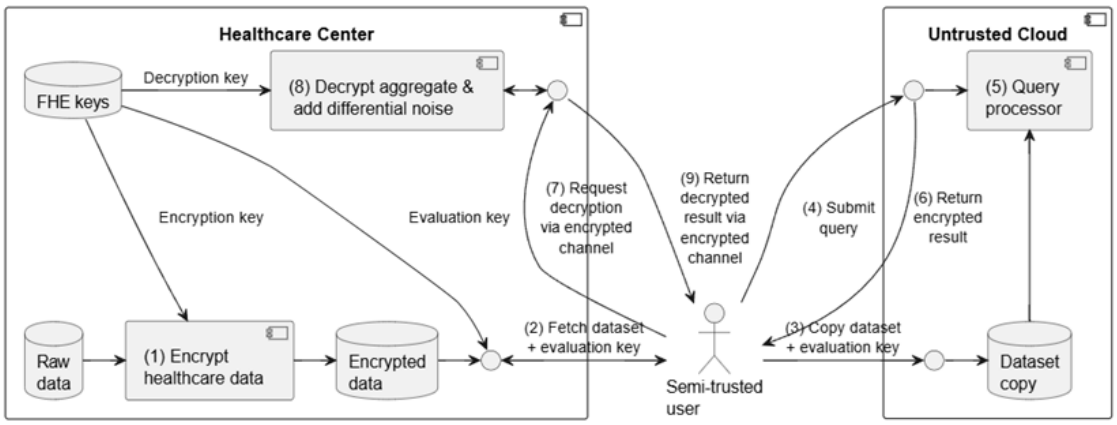

Figure 3.1: Process schema for the Scenario S_4 which uses both DP and FHE. The healthcare center wishes to share D with a semi-trusted user for analysis. Since the user may only see aggregates, D is first homomorphically encrypted to produce $E(D)$, and then shared with the user together with its evaluation key K_e. The user copies over K_e and $E(D)$ to an untrusted environment. The user then performs queries that are homomorphically computed in the untrusted environment, yielding a result $E(R)$. The user requests decryption of $E(R)$ to the healthcare center, which responds with $R + e$ after decryption, where e is noise added by the Laplace mechanism.

3.3 Experimental setup

This section describes how the testbed is set up and what execution environment is used. It also describes the evaluation criteria and how they are measured.

3.3.1 Testbed setup and execution environment

The process described in Figure 3.1 is implemented as Docker images that are deployed in a Kubernetes cluster, communicating with each other using a REST interface. The Kubernetes setup is a one-node setup run on a host with 32 GB RAM and an 8-core CPU with 16 threads and a 4.2 GHz clock speed.

The resulting experimental setup of this project is described in Figure 3.2. Three different containers run on the experimental computing environment: the user, the healthcare center, and the untrusted cloud. The user communicates with the untrusted cloud and the healthcare center via network requests. The healthcare center never directly communicates with the user's untrusted cloud.

Each experiment starts with the healthcare center container generating keys (and compiling the circuit in the case of Concrete) based on the encryption parameters, and encrypting the datasets. In this simplified setup, the file system is shared between the healthcare center container and untrusted cloud container, although the untrusted cloud container never accesses the private key. In a real setup, there would be full isolation and no communication between the untrusted cloud and the healthcare setup. After this pre-processing step, the user container queries

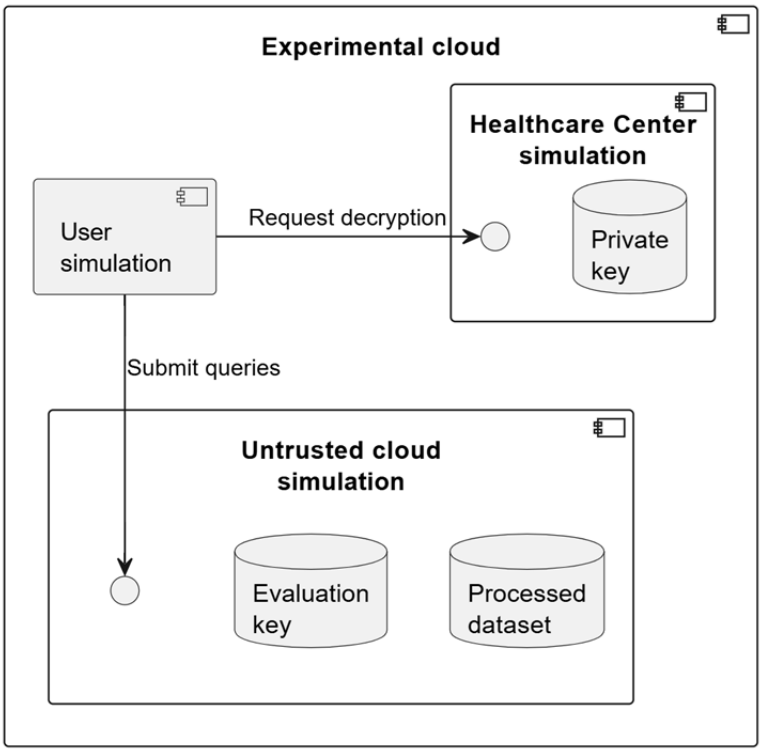

Figure 3.2: Experimental setup. The testbed is realized as three Docker containers running in a cloud environment. The containers represent the untrusted cloud environment, the user's environment and the healthcare center's environment. The arrows represent the data that is shared between the containers. A simplification has been done compared to the process schema in Figure 3.1. Instead of implementing the simulation of the user moving the dataset from the Healthcare center to the untrusted cloud environment, the untrusted cloud is already provided with the dataset before simulation begins. This simplification does not affect the evaluation criteria of the testbed.

the untrusted cloud container for evaluation of the given query and scheme. The untrusted cloud container then evaluates the chosen query homomorphically and returns the result and evaluation metrics to the user container. Finally, the user container sends a query for decryption to the healthcare center container, which returns the decrypted value of the homomorphic evaluation, along with additional metrics. Finally, the user container collects the metrics produced by the other containers and writes the results to a CSV file.

Each experiment records metrics used for evaluation, which include evaluation time, ciphertext expansion, dataset size, chosen scheme and query, and chosen encryption parameters. Each experiments contain multiple runs, which produce records that is saved in a separate dataset containing results of all experiments. This dataset is then analyzed to produce the final evaluation metrics.

For the baseline experiment without encryption and differential privacy, we simply create a dataset containing the query result, dataset size, and evaluation time with-

out simulating the experiment in a containerized environment. This does not affect the evaluation criteria, but provides a baseline against which other configurations can be evaluated.

For scenarios with Differential Privacy, we apply the noise separately on the results dataset, to show what the results would have been in case differential privacy had been used. This reduces the amount of experiments that need to be done and simplifies evaluation by making the application of noise independent from the homomorphic evaluation, and does not affect the evaluation metrics.

3.3.2 Evaluation criteria

The testbed is evaluated through the queries f_1, f_2 and f_3, described in Section 3.2.4 on the dataset. Moreover, the encrypted datasets themselves is evaluated according to their ciphertext expansion and accuracy upon decryption. The evaluation criteria is presented in Table 3.3.

Criterion	**Metric**	**Description**
Evaluation time	seconds	Time to evaluate query homomorphically
Accuracy	MAE	Mean absolute error
Accuracy	MSE	Mean squared error
Noise magnitude factor	scalar	Factor of absolute noise from expected result
Ciphertext size	bytes	Size of evaluated ciphertexts
Ciphertext expansion	scalar	Factor of expansion from plaintext

Table 3.3: Evaluation criteria of the testbed for execution of each query.

3.3.3 Query design

Implementing circuits for homomorphic evaluation comes with many different challenges. A lot of care needs to be taken when choosing parameters, strategy for evaluation, managing noise and bit-width, and so on. Also, the libraries used in this project, Concrete (for TFHE) and SEAL (for BGV, BFV, and CKKS) use considerably different approaches for realizing homomorphic evaluation, as shall be seen.

As implementing the aggregate queries described in Section 3.2.4 is not as straight forward as it might look at first glance, this Section explains to describe the design choices for each query.

For both Concrete and SEAL, we instantiate the fields in the datasets as NumPy arrays. However, the interface differs drastically between SEAL and Concrete. In Concrete, evaluation circuits are defined in a regular function, which is then compiled into a homomorphic evaluation circuit. In SEAL, an interface is provided for piecewise evaluations (e.g. a function for homomorphic addition, subtraction, multiplication, etc.). The circuit end-to-end is written by the developer by chaining together piecewise evaluations.

3.3.3.1 Query 1: sum of expenses

The mathematical formulation for this query is

$$f_1(exp) = \sum_{i}^{n} exp_i.$$

Concrete implements certain NumPy aggregations out-of-the-box, so in Concrete, simply calling .SUM() on the expenses array will create a circuits that sums the array elements and returns the total sum.

In SEAL, the circuit for evaluating the sum of an array is created with a homomorphic vector sum algorithm, showed in Algorithm 2.

Algorithm 2
Array sum algorithm in SEAL [45]. It uses RotateRows() provided by the Evaluator class in SEAL, which shifts the position of the Array items by one.

Input: Encryption scheme *scheme*, evaluator E, Galois key K_g, ciphertext c, dataset size n
Output: Ciphertext containing aggregated result

function SUM($scheme, E, K_g, c$):
  $c' \leftarrow c$
  $step \leftarrow 1$
  while $step < n$ **do**
    $rotated \leftarrow E$.ROTATEROWS($result, step, K_g$)
    $c' \leftarrow E$.ADD($result, rotated$)
    $step \leftarrow step * 2$
  return c'

3.3.3.2 Query 2: average income after expenses

This query is mathematically formulated as

$$f_2(inc, exp, cov) = \frac{1}{n}(\sum_{i}^{n} inc_i + \sum_{i}^{n} cov_i - \sum_{i}^{n} exp_i).$$

Similar to the first construct, but contains division with unencrypted constant.

In this query, we see the distinction between different generations of schemes. Namely, the schemes BFV and BGV do not support either homomorphic division or floating points. In TFHE, this operation is supported with a homomorphic Lookup Table (LUT). In CKKS, division with unencrypted denominator is supported by multiplicating with the multiplicative inverse of the denominator. Since CKKS supports floating point operations, multiplication can be done with values between 0 and 1. Since BFV and BGV support neither LUTs nor floating point operations, division is not supported.

Therefore, we implement this query in two different ways to enable comparison between all schemes. The first method is to have the healthcare center container

do the division *after* decryption. This means that the full circuit is not calculated homomorphically, but this approach supports all schemes. The second method is to evaluate the full circuit homomorphically. This approach only works for the TFHE and CKKS schemes because of the aforementioned limitations of the BGV and BFV.

In TFHE, division is implemented with LUTs, but these only enable support for division with a 16-bit numerator in Concrete (a positive integer with a maximum value of 255). Since the sum of income plus coverage minus expenses obviously exceeds 16 bits by several orders of magnitude, this sum needs to be packed into a 16-bit integer before it can be evaluated homomorphically. In this project, we achieve this by, prior to encryption, letting the healthcare center container run the equivalent query non-homomorphically, counting the bits of the query result, and shifting by the number of bits required to make the sum fit in 16-bits. We then shift all records in the dataset with the same amount of bits before encrypting it and running the experiment. At the end of the experiment, we shift back to the original bit-width to get the actual value of the decrypted result.

For CKKS, the implementation is much simpler due to floating point support. We calculate the multiplicative inverse of the dataset size in plaintext, encode it and multiply the sum with the inverse.

3.3.3.3 Query 3: average coverage

This query is formulated with

$$f_3(exp, cov) = \frac{\sum_i^n cov_i}{\sum_i^n exp_i + \sum_i^n cov_i}.$$

Unlike to Query 2, this query has both an encrypted numerator and an encrypted denominator. The consequences of this is that neither a LUT can be used (as in the case of TFHE), and a multiplicative inverse cannot be calculated since division is not natively supported in the CKKS scheme. This leads to the question whether it is possible to approximate the quotient homomorphically, using operations which are supported in TFHE and CKKS. In Concrete, it is possible to create a circuit which encodes a value between 0 and 1 to a 16-bit integer, which can then be decoded after decryption. However, an equivalent circuit is seemingly not possible to implement in SEAL with CKKS due to the lack of support for comparative operations.

Seeing as it was seemingly not possible to implement the circuit in SEAL, we opted for an approach where the user container sends the encrypted operands separately to the healthcare center container. The healthcare center container then decrypts the numerator and denominator separately and returns the quotient. This approach can be evaluated with all schemes.

3.3.4 Query sensitivity and noise

The Laplace mechanism was applied to the results of the previous experiment to show what the results would have been with Differential Privacy. For these experiments, we first calculate the L_1 sensitivity of the queries, which is the maximum value a

single individual can have on the output of the query. As there is no closed form equation that can determine the sensitivity, we determine the sensitivities in the following ways:

- **Query 1:** Here, for the query $f_1(exp) = sum(exp)$, we use the maximum value of the expenses field in the dataset as the sensitivity; $\Delta f_1 = max(exp)$. The expenses of a single individual can not exceed this value.
- **Query 2:** For the query

$$f_2(inc, cov, exp) = \frac{1}{n}\Big(\sum_i^n inc_i + \sum_i^n cov_i - \sum_i^n exp_i\Big),$$

  we use the sensitivity $\Delta f_2 = \frac{1}{n}[max(inc) + max(cov) - min(exp)]$. Here, we approximate n to the actual dataset size.
- **Query 3:** For the query

$$f_3(cov, exp) = \frac{\sum_i^n cov}{\sum_i^n exp + \sum_i^n cov},$$

  we could use the sensitivity $\Delta f_3 = 0.5/n$, which is the maximum sway of the result possible for one record, and is achieved if a person has full coverage of expenses.

For Query 2, there are different sensitivities depending on the size of the dataset. As described in Section 3.2, we use three different dataset sizes 10^2, 10^3 and 10^4, and these values are used to set the sensitivity of the query. The resulting L_1-sensitivities are displayed in Table 3.4.

L_1-sensitivity	Value
Query 1: Δf_1	845363
Query 2: $\Delta f_{2,100}$	24591
Query 2: $\Delta f_{2,1000}$	2459
Query 2: $\Delta f_{2,10000}$	246
Query 3: $\Delta f_{3,100}$	5*10−3
Query 3: $\Delta f_{3,1000}$	5*10−4
Query 3: $\Delta f_{3,10000}$	5*10−5

Table 3.4: L_1-sensitivities used for the Laplace mechanism in this project. Note that for query 2, the sensitivity differs according to dataset size.

Finally, the privacy budgets considered for this experiment are $\epsilon = 10$, $epsilon = 5$, $epsilon = 1$, $epsilon = 0.5$, and $epsilon = 0.1$. This represents commonly chosen privacy budgets in literature.

3.3.5 Encryption parameter selection

A big difference between Concrete and SEAL shows up already when selecting the parameters. Concrete does not allow parameters to be set directly, and instead uses

an internal parameter selection algorithm based on a strategy chosen by the user. Two strategies are available, MONO and MULTI. Concrete implements a parameter search strategy to find the optimal parameters for the specified circuit [65]. The tradeoff is that MULTI executes circuits faster, but leads to slower key generation, larger keys, and larger memory usage. Since there is no reason to believe that memory or storage space is a significant constraint in a cloud environment, we prioritize execution time and select MULTI for evaluating the TFHE scheme.

SEAL, on the other hand, allows selecting parameters directly, and a validator ensures validity of the chosen parameters upon instantiation. The parameters tuned for optimization is shown in Table 3.5.

Library	**Scheme**	**Parameter**
Concrete	TFHE	Parameter selection strategy
SEAL	BFV, BGV, CKKS	Polynomial modulus degree
SEAL	BFV, BGV, CKKS	Coefficient modulus
SEAL	BFV, BGV	Plaintext modulus

Table 3.5: Parameters exposed to library users for fine-tuning performance. Concrete does not allow direct fine tuning, and instead tunes the parameters based on chosen strategy. SEAL allows direct fine tuning, but it is up to the user to set correct parameters.

Firstly, we set the polynomial modulus degree to a power of two, which is required performance and security reasons. A lower poly modulus degree increases the noise budget, but leads to worse performance [45]. As such, for the best performance, the lowest polynomial modulus degree that does not exceed the noise budget for the circuit should be chosen. We therefore select the polynomial modulus degrees 8096, 8096, and 32768 for the small, medium, and large datasets, respectively.

The optimal setting for the plaintext modulus depends on the maximum value of the evaluation circuit and the poly modulus degree. A lower setting is desirable, but if the number of bits in the evaluation circuit exceeds the number of bits needed to evaluate the query, an integer overflow occurs and subsequent decryption leads to an incorrect result. The optimal number of bits is chosen for BGV and BFV is displayed in Table 3.6.

	$n = 100$	$n = 1000$	$n = 10000$
Q1	26	29	33
Q2	27	30	34
Q3	26	29	32

Table 3.6: Number of bits used for per query and dataset size using the PlainModulus.Batching class in SEAL-Python. This class calculates the plain modulus using polynomial modulus degree and batching bits.

Finally, the optimal setting for coefficient modulus depends on whether the scheme is CKKS, or BFV and BGV. For this parameter, we used the default value for

all schemes. A class called BFVDefault can be used for setting a recommended coefficient modulus based on the polynomial modulus degree for the BFV and BGV schemes. For CKKS, we used CoeffModulus provided by SEAL to create a coefficient modulus based on the polynomial modulus degree and prime bit sizes of 60, 40, 40, and 60 bits.

4

Results

In this Chapter, the results of the project are presented. First, the compliance of scenarios to requirements are stated. Second, we compare the different scenarios on the selected evaluation criteria.

4.1 Requirements compliance

In Section 3.1, four different scenarios were posed using different combinations of Fully Homomorphic Encryption (FHE) and Differential Privacy (DP). Scenario S_1 features neither DP nor FHE, S_2 features only DP, S_3 features only FHE, and S_4 features both DP and FHE. The compliance of each scenario to the requirements is shown in Table 4.1. The requirements can be found in Section 3.2.5.

	S_1	S_2	S_3	S_4
REQ1	Y	Y	Y	Y
REQ2	Y	Y	Y	Y
REQ3	N	N	Y	Y
REQ4	N	Y	N	Y

Table 4.1: Comparison of requirement compliance for different scenarios. All scenarios comply with the availability requirements REQ1 and REQ2, whereas Scenario 1, 2, and 3 do not comply with both confidentiality requirements REQ3 and REQ4.

Scenario 4 complies with all stated requirements, whereas Scenario 2 and 3 fails to comply with the all of the stated confidentiality requirements. Finally, Scenario 1 fails to comply with any of the confidentiality requirements, and only succeeds to comply with the availability requirements.

4.2 Implementation of queries

This Section described whether the queries were implemented. Recall the definition of the Queries from Section 3.2.4:

- **Query 1:** $f_1(D) = \sum_i^n exp_i$
- **Query 2 (simplified):** $f_2(D) = \sum_i^n (inc_i + cov_i - exp_i)/n$, division step done after decryption

- **Query 2 (with division):** $f_2(D) = \sum_i^n (inc_i + cov_i - exp_i)/n$, division step done homomorphically
- **Query 3:** $f_3(D) = \frac{\sum_i^n cov_i}{\sum_i^n (cov_i + exp_i)}$, division step done after decryption

In Table 4.2, it is shown which scheme could implement which queries. The division operation is supported in TFHE and CKKS, but not BGV and BFV. Therefore, two variants were created to be able to compare both the simplified and the query with division.

	BFV	BGV	TFHE	CKKS
Query 1	Y	Y	Y	Y
Query 2 (simplified)	Y	Y	Y	Y
Query 2 (with division)	N	N	Y	Y
Query 3	Y	Y	Y	Y

Table 4.2: Whether queries were implemented. For Query 2, the BFV and BGV schemes do not support division, and SEAL provides no interface for approximation. Therefore, two variants were created to be able to compare the schemes.

4.3 Evaluation time

Each experiment, which consists of homomorphically evaluating encrypted operands for the chosen query and scheme, was conducted 15 times to attain the mean evaluation time. The results are displayed in Figure 4.1.

Recall that Query 2 was implemented in two ways, one where the division is computed homomorphically by the untrusted cloud container, which is supported only by the TFHE and CKKS schemes, and one where the division is computed after decryption by the healthcare center container. No results are shown for BGV and BFV on Query 2 with division as the division operation is not supported in these schemes.

In all queries, with the exception of Query 2 with division, the TFHE scheme performed better. For Query 2 with division, the TFHE scheme (using 16-bit LUTs to evaluate division) performed worse than the CKKS scheme (using multiplicative inverse to evaluate division) by a factor of approximately 10^3.

The fastest evaluation time in the Graph is Query 1 with TFHE, which takes approximately 18 ms with a dataset size of 10000. The equivalent computation in plaintext was 11 μs, approximately 150 times faster.

4.4 Accuracy

There are two aspects that affect the accuracy of the aggregate queries. First, in Scenario S_3 and S_4, accuracy is lost while undergoing homomorphic processing (en-

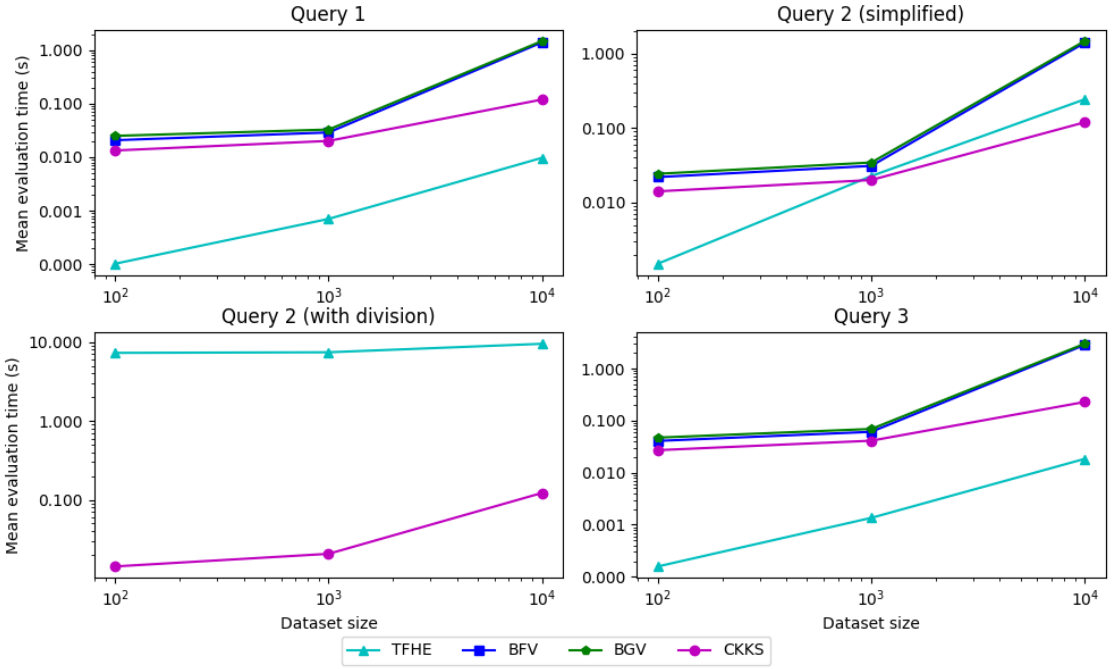

Figure 4.1: Evaluation times of schemes. Each query and scheme was executed 15 times to attain the mean evaluation time. The scales are logarithmic on both the x and the y-axes.

cryption, evaluation, and decryption). Second, in Scenario S_2 and S_4, Laplace mechanism is applied to the output of the query, which perturbs the result further.

4.4.1 Homomorphic circuits

For experiments in scenarios with FHE, Scenario S_3 and S_4, FHE typically does not introduce additional noise to the output. However, there are some exceptions; experiments using the CKKS scheme and experiments executing Query 2 in TFHE. The noise introduced by homomorphic evaluation after decryption is shown in Figure 4.2.

The imprecision for the CKKS scheme does not necessarily scale with dataset size; sometimes there is a positive correlation, but at other times there is a negative correlation. The error is typically very small as a magnitude compared to the expected result, except for the case of TFHE in Query 2, where the deviation is significant. The implementation of Query 2 is explained in further detail in Section 3.3.3.2.

To place the result of these experiments in context, it is helpful to express the error as a factor of the expected output of the query. This factor is shown in Figure 4.3 for the different queries and dataset sizes.

4.4.2 Laplace mechanism

For experiments in scenarios with Differential Privacy (i.e. Scenario S_2 and S_4), the accuracy of the query outputs depends on the L_1-sensitivity and the privacy budget. Furthermore, for Query 2, the accuracy also depends on the dataset size, which is

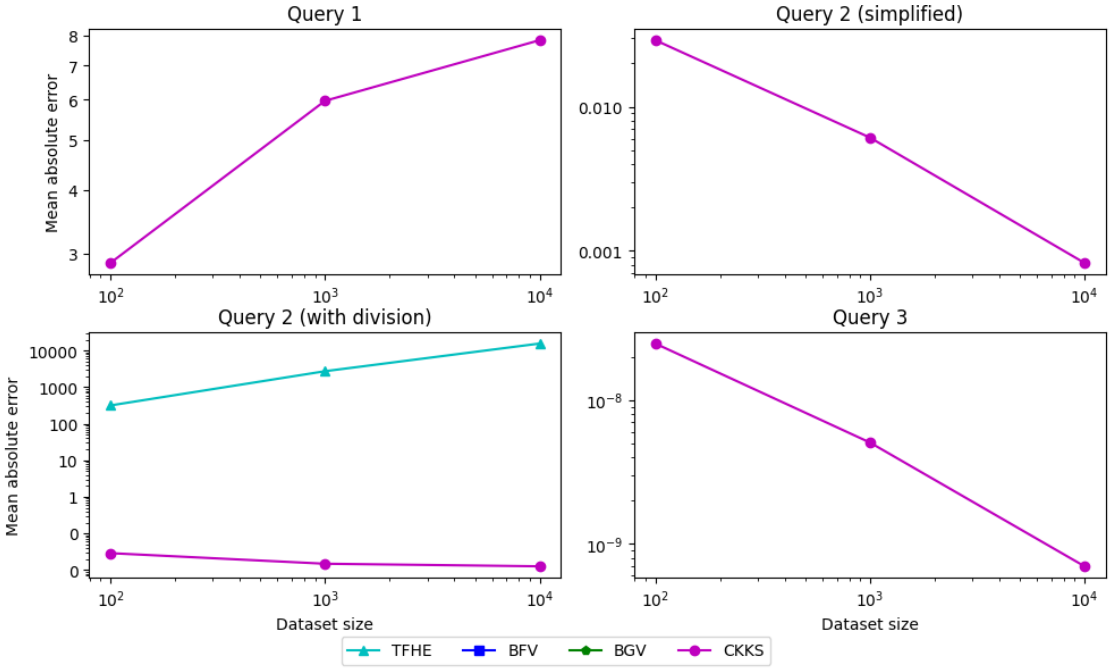

Figure 4.2: Mean absolute error of query outputs. The mean absolute error is zero for legend items that are not shown in the plots, except for Query 2 with division, where there are no results for BFV and BGV schemes.

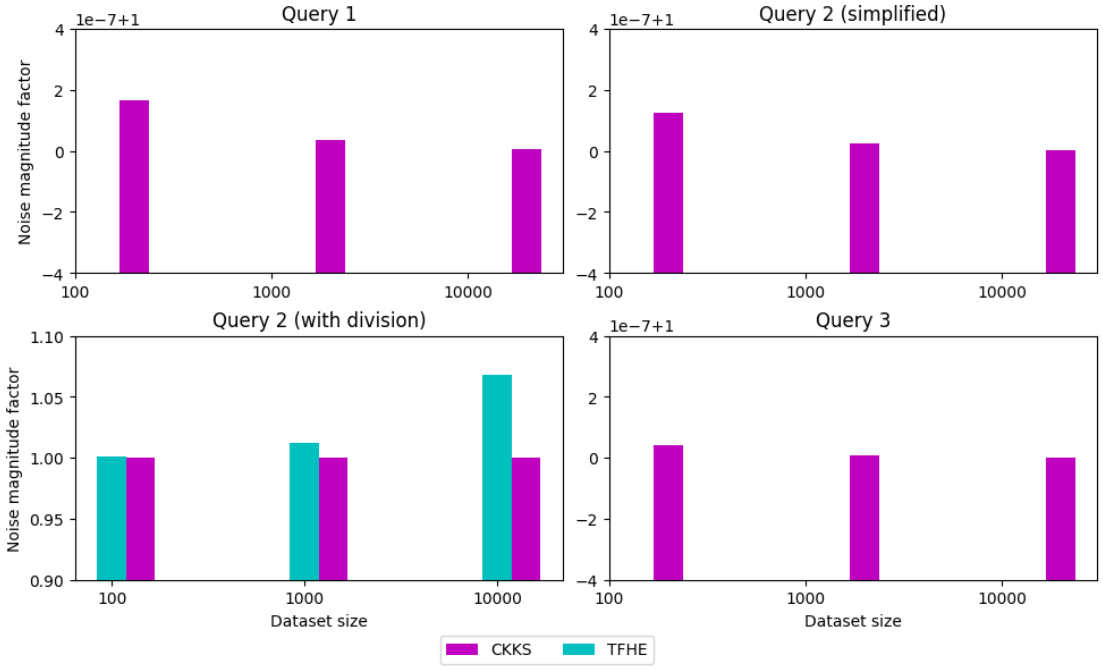

Figure 4.3: Magnitude of noise as factor of the expected result. The noise for the CKKS scheme, the maximum factor observed in these plots is approximately 0.0002%. For TFHE in Query 2, this factor is closer to 7%.

either 10^2, 10^3 or 10^4 in this experiment. The MAE and MSE metrics when applying noise are displayed in Figure 4.4. The MAE shows the average magnitude of the

additional noise that is applied on top of the query result.

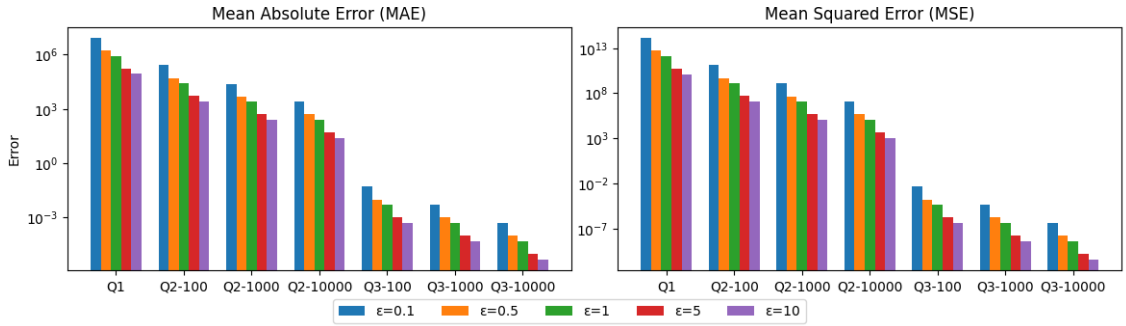

Figure 4.4: Mean Absolute Error (MAE) and Mean Squared Error (MSE) of the Laplace mechanism for the three queries and dataset sizes under different L_1-sensitivities and privacy budgets (ϵ). MAE shows the average magnitude of the additional noise that is applied to the query result, and MSE is the squared error to punish outliers. Note that for Query 2, the sensitivity is different based on the size of the dataset. The sample size is $n = 1000$ per experiment.

The factor of the noise added to the output compared to the expected output is expressed in Figure 4.5. For CKKS, the additional noise added to the query output is significantly smaller than the noise added by the Laplace mechanism. For TFHE in Query 2, the noise added after decryption can exceed the noise added with the Laplace mechanism under higher privacy budgets.

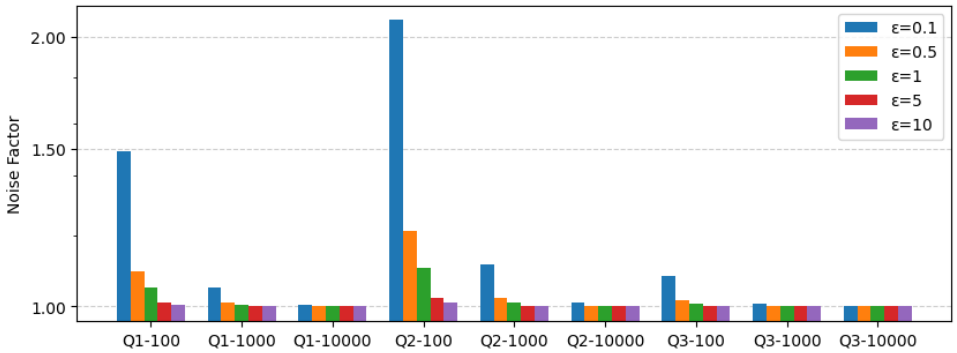

Figure 4.5: Magnitude of noise as factor of the expected result for Laplace mechanism for the different queries, dataset sizes and privacy budgets (ϵ).

4.5 Ciphertext expansion

In Figure 4.6, the ciphertext size after evaluation is shown. The CKKS scheme had the lowest ciphertext expansions, especially with the large dataset. The plaintext size occupies up to 4 bytes, so the ciphertext expands by several orders of magnitude.

It should be noted that Concrete development API lacks direct access to the ciphertext, instead only providing it in a serialized format for sending keys via the network. This serialized format is seemingly compressed, which is why the ciphertext size cannot be compared to the ciphertext sizes of the CKKS, BFV, and BGV

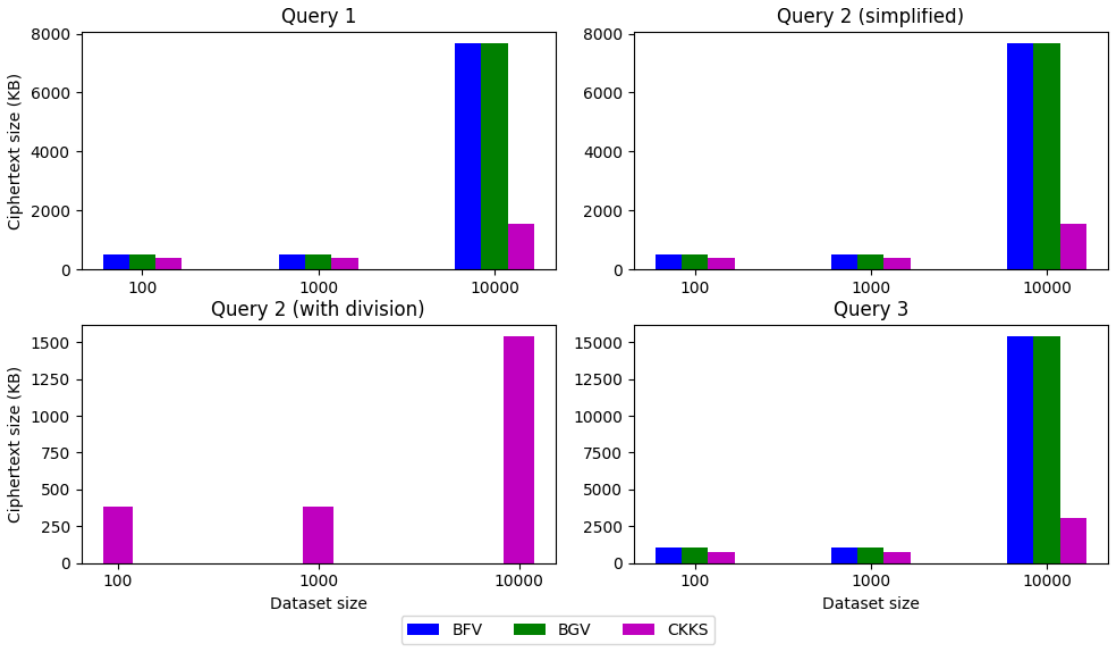

Figure 4.6: Evaluated ciphertext sizes in kilobytes. For reference, the plaintext occupies up to 4 bytes of space. The BGV and BFV schemes have the same ciphertext sizes, whereas the CKKS scheme scales better with higher dataset sizes.

schemes. Therefore, the results for the TFHE scheme have been omitted in these experiments.

5

Discussion

In this section, we discuss the execution and methodology. The security aspects are discussed according to the threat model where the user is honest-but-curious (i.e. wishes to reveal the underlying information, but otherwise has legitimate reasons to process the data), and the untrusted cloud provider is malicious and wishes to gain illegitimate access to the data.

5.0.1 Requirement compliance

In the methodology, four different scenarios were posed, containing combinations of Fully Homomorphic Encryption (FHE) and Differential Privacy (DP), or lack thereof. Four requirements were stated, and two of these were patient data confidentiality requirements while the other requirements were availability requirements. While all scenarios achieve the availability requirements, only the scenario that combines FHE and DP achieves both confidentiality requirements.

In Scenario S_1 and S_2, the data is not encrypted. This means that the data can either not be made available to a trusted user, thus breaking the availability requirements, or the data is made available to the trusted user, but in an unencrypted format, thus breaking the confidentiality requirements. In Scenario S_3, the data is available and encrypted for the trusted user, but the trusted user can perform membership inference attacks. In Scenario S_4, patients' data are protected from membership inference by the honest-but-curious user and eavesdropping by the untrusted cloud owner, while still being available for processing with homomorphic encryption.

However, there are threats to the confidentiality requirements. First, the ciphertext set remains the same before and after evaluation. As a result, the honest-but-curious user can request a decryption of an entry in the dataset, without performing any evaluations. Since the additional noise is added based on the sensitivity of the query, the wrong sensitivity is used in the Laplace mechanism when decrypting data that has not been homomorphically evaluated. A proposal to mitigate this issue is to require the user to provide a Zero-Knowledge Proof prior to decryption that the ciphertext is indeed the result of a homomorphic evaluation. This is called *verifiable homomorphic encryption* [66].

Second, assuming that the untrusted cloud provider has access to the hardware where homomorphic evaluations are executed on, a wide number of side-channel attacks may become feasible, including measuring power consumption, timing, and

electromagnetic radiation [67]. Furthermore, since the cloud provider has access to ciphertext inputs, outputs, evaluation keys, and circuits, it can freely perform side-channel attacks in an isolated environment by evaluating different inputs. Therefore, deploying Fully Homomorphic Evaluation circuits on an untrusted server, while safe from a theoretical standpoint, requires deeper understanding of risks regarding side-channel attacks. In fact, while not a vulnerability in the experimental setup, a side channel attack has previously been discovered in a previous version of the SEAL library which reveals the plaintext during the encryption phase with a power measurement [68]. However, more research is needed in this area to understand the security of executing homomorphic evaluations in an insecure environment.

5.0.2 Performance

The results of this project shows that TFHE has the best performance regarding execution speed. This is unexpected, because another comparative study demonstrated that CKKS performs better than TFHE [69]. The cause of this could be that this study used HELib instead of SEAL to measure the performance of CKKS. Another cause could be that the hardware used in our experiments differ significantly from the hardware used in that paper.

Combining FHE and DP in a privacy-preserving system comes with drawbacks in performance and accuracy. FHE schemes are typically constrained by circuit depth, where deep circuits cause execution time to increase rapidly. Since analysis on big data typically operates on large datasets, this often leads to deep circuits and slow execution times. It is therefore impractical to execute aggregate queries on large datasets homomorphically.

On the other hand, the usefulness of DP depends on having a sufficiently large dataset, so that a single record does not perturb the query result by too large a factor. Small datasets provide inaccurate results with the privacy budget chosen in this project, whereas large datasets become impractical to process homomorphically. Therefore, FHE and DP works against each other in a sense, and for the combination DP and FHE to be useful in the setting outlined by this project, performance improvements are needed for FHE.

Ciphertext expansion is also a concern. Although the CKKS scheme performed better than BFV and BGV, ciphertext sizes still reached several megabytes, a significantly bigger size than the original plaintext size. Compressing ciphertexts before storing them or sending them over the network will mitigate ciphertext expansion, but not eliminate the issue.

5.0.3 Usability

Aside from performance, there are usability and practical concerns when implementing FHE. There are design considerations when assessing what should be evaluated homomorphically and what should be computed after decryption. For example, Query 2 was implemented in two ways; one where the entire average was calculated homomorphically and one where the final division took place after the rest

of the query had been executed. Both ways fulfill confidentiality and availability requirements, but the simplified version performs better.

TFHE and CKKS could be used to implement more advanced queries. As the CKKS scheme allows floating point numbers, it has more applications in data analysis. For example, the evaluation time of Query 2 in CKKS was significantly smaller due to the possibility to use multiplicative inverse instead of division. In addition, in TFHE, the division operation was forced to be implemented with LUTs, which only support up to 16-bit operands. Because of the 16-bit limitation, the inaccuracy of the result grows as the dataset size increases, since more of the least significant bits are lost.

Another consideration is the decision on which library to use when implementing FHE. SEAL and Concrete used two different approaches, with Concrete letting the user define the entire circuit end-to-end, whereas in SEAL, the user chains together piecewise evaluation circuits like additions and multiplications. The circuit pre-compilation step in Concrete could be a factor to why the TFHE scheme performed better in the evaluation speed, especially since this enables Concrete to find the optimal parameters. Although it should be noted that with pre-compilation of circuits, it is a challenge to write circuits that do not exceed noise budgets or have unsupported operations. From a data analysis perspective, CKKS is likely the most usable scheme due to the support for floating point operations, but SEAL lacks some features that could be usable for CKKS, such as bootstrapping [69].

For Differential Privacy, the design choice is to choose as low as a sensitivity as possible. For simple queries, it is easy to estimate the sensitivity, but due to the lack of a closed-form equation, more complex queries are harder to estimate the sensitivity of. Additionally, due to the statistical nature of DP, it is challenging to estimate the impacts and risks of a chosen privacy budget, since there is a lack of methods and tools to analyze the query result.

5.0.4 Ethical considerations

Since this project deals with computation of private data, although de-identified and encrypted, it is important to consider the risks when handling private data.

Modern cryptography is founded on the principle that, without access to the secret key, it is computationally infeasible to retrieve the original plaintext from the ciphertext. This principle relies on the assumption that the underlying mathematical problem for the cryptographic scheme can only be efficiently solved by the secret key holder. However, this assumption may not be true, as there might exist attacks against the cryptographic scheme that nobody has yet discovered. However, new attacks can be discovered over time, such as a successful chosen plaintext attack which recovered the secret key has been demonstrated on the CKKS scheme [42].

Data processors are also required to abide by data privacy regulations. Regulations in United States such as Health Insurance Portability and Accountability Act (HIPAA) demand a written authorization report from patients when it comes to sharing private health data with external parties [70]. Whereas European regulations

such as General Data Protection Regulation (GDPR) impose limitations on how data is stored, processed, and shared; non-compliance to those regulations may lead to substantial fines [71]. Due to the nature of the patient data, the real healthcare data can only be shared between the authorized users such as physicians, database administrators of the healthcare center. Therefore, organizations who wish to use the method in this project must investigate the legal implications of releasing data to the public, even if it is de-identified and encrypted.

6

Conclusion

The goal of this project was to create a privacy preserving system that combined Differential Privacy and Fully Homomorphic Encryption to explore the effects they have on performance, usability and privacy. The Research Questions posed in Chapter 1 were as follows:

- **RQ1:** How does combining FHE and DP affect the privacy guarantees of data subjects?
- **RQ2:** What potential vulnerabilities exist in a system combining FHE and DP, and under which conditions can these vulnerabilities enable re-identification of data subjects?
- **RQ3:** How does the performance of encrypted aggregation queries compare to non-encrypted aggregation queries?
- **RQ4:** How do different FHE schemes and libraries influence performance according to the defined evaluation criteria?

For RQ1, there are no synergistic effects of combining DP and FHE, but each method had an effect on its own. With FHE, it was possible to process encrypted data without decrypting it, which provides stronger guarantees in certain cloud environments. With DP, it was possible to protect the individuals from having their data leaked in aggregate queries.

For RQ2, processing data in an unsafe environment comes with some risks. First, processing ciphertexts homomorphically in insecure environments is risky as it potentially provides an adversary with the possibility of attempting side-channel attacks such as power measurements or timing attacks to recover plaintexts from ciphertexts. Second, the decryption component must ensure that the correct L_1-sensitivity is always applied to the query output, otherwise the privacy guarantee falters and a membership inference attack can be carried out.

For RQ3, homomorphic evaluations are at least 150 times slower than the equivalent plaintext computation. Homomorphically evaluated ciphertexts expand to orders of magnitude bigger than the encrypted plaintext. The evaluation times and ciphertext sizes increase with larger aggregations.

For RQ4, it was observed that the TFHE scheme had the best performance according to the evaluation criteria. However, there is reason to doubt this result, as other

papers have shown CKKS to be more performant. The cause for this discrepancy could be the choice of FHE library or experimental hardware. Furthermore, CKKS is more usable for statistical analysis thanks to the support for floating point numbers.

This thesis demonstrated that while privacy guarantees improve with DP and FHE, practical and performance concerns remains big challenges in FHE. In its current form, the system is not yet suitable for large-scale deployment. Feasibility of large-scale deployments requires significant improvements in FHE performance and broader support for commonly used statistical operations in the FHE ecosystem.

Bibliography

[1] J. L. Raisaro et al., "Medco: Enabling secure and privacy-preserving exploration of distributed clinical and genomic data," *IEEE/ACM transactions on computational biology and bioinformatics*, vol. 16, no. 4, pp. 1328–1341, 2018.

[2] W. Khan et al., "Person de-identification: A comprehensive review of methods, datasets, applications, and ethical aspects along-with new dimensions," *IEEE Transactions on Biometrics, Behavior, and Identity Science*, 2024.

[3] J. L. Raisaro et al., "Protecting privacy and security of genomic data in i2b2 with homomorphic encryption and differential privacy," *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, vol. 15, no. 5, pp. 1413–1426, 2018. DOI: 10.1109/TCBB.2018.2854782.

[4] D. Kim, G. Lee, and S. Oh, "Toward privacy-preserving text embedding similarity with homomorphic encryption," in *Proceedings of the Fourth Workshop on Financial Technology and Natural Language Processing (FinNLP)*, 2022, pp. 25–36.

[5] R. Aziz, S. Banerjee, S. Bouzefrane, and T. Le Vinh, "Exploring homomorphic encryption and differential privacy techniques towards secure federated learning paradigm," *Future internet*, vol. 15, no. 9, p. 310, 2023.

[6] J. H. Cheon, A. Kim, M. Kim, and Y. Song, "Homomorphic encryption for arithmetic of approximate numbers," in *International conference on the theory and application of cryptology and information security*, Springer, 2017, pp. 409–437.

[7] A. Al Badawi, L. Hoang, C. F. Mun, K. Laine, and K. M. M. Aung, "Privft: Private and fast text classification with homomorphic encryption," *IEEE Access*, vol. 8, pp. 226 544–226 556, 2020.

[8] W. Diffie and M. E. Hellman, "Multiuser cryptographic techniques," in *Proceedings of the June 7-10, 1976, national computer conference and exposition*, 1976, pp. 109–112.

[9] R. L. Rivest, "Cryptography," in *Algorithms and complexity*, Elsevier, 1990, pp. 717–755.

[10] T. ElGamal, "A public key cryptosystem and a signature scheme based on discrete logarithms," *IEEE transactions on information theory*, vol. 31, no. 4, pp. 469–472, 1985.

[11] P. Martins, L. Sousa, and A. Mariano, "A survey on fully homomorphic encryption: An engineering perspective," *ACM Computing Surveys (CSUR)*, vol. 50, no. 6, pp. 1–33, 2017.

[12] R. L. Rivest, L. Adleman, M. L. Dertouzos, et al., "On data banks and privacy homomorphisms," *Foundations of secure computation*, vol. 4, no. 11, pp. 169–180, 1978.

[13] C. Gentry, "Fully homomorphic encryption using ideal lattices," in *Proceedings of the forty-first annual ACM symposium on Theory of computing*, 2009, pp. 169–178.

[14] I. Chillotti, N. Gama, M. Georgieva, and M. Izabachène, "Tfhe: Fast fully homomorphic encryption over the torus," *Journal of Cryptology*, vol. 33, no. 1, pp. 34–91, 2020.

[15] F. Armknecht et al., "A guide to fully homomorphic encryption," *Cryptology ePrint Archive*, 2015.

[16] A. Acar, H. Aksu, A. S. Uluagac, and M. Conti, "A survey on homomorphic encryption schemes: Theory and implementation," *ACM Computing Surveys (Csur)*, vol. 51, no. 4, pp. 1–35, 2018.

[17] Y. Doröz and B. Sunar, "Flattening ntru for evaluation key free homomorphic encryption," *Journal of Mathematical Cryptology*, vol. 14, no. 1, pp. 66–83, 2020.

[18] C. Gentry, *A fully homomorphic encryption scheme*. Stanford university, 2009.

[19] Z. Brakerski, C. Gentry, and V. Vaikuntanathan, "(leveled) fully homomorphic encryption without bootstrapping," *ACM Transactions on Computation Theory (TOCT)*, vol. 6, no. 3, pp. 1–36, 2014.

[20] C. Gentry and S. Halevi, "Implementing gentrys fully-homomorphic encryption scheme," in *Annual international conference on the theory and applications of cryptographic techniques*, Springer, 2011, pp. 129–148.

[21] O. Regev, "On lattices, learning with errors, random linear codes, and cryptography," *Journal of the ACM (JACM)*, vol. 56, no. 6, pp. 1–40, 2009.

[22] O. Regev, "The learning with errors problem," *Invited survey in CCC*, vol. 7, no. 30, p. 11, 2010.

[23] N. Bindel, J. Buchmann, F. Göpfert, and M. Schmidt, "Estimation of the hardness of the learning with errors problem with a restricted number of samples," *Journal of Mathematical Cryptology*, vol. 13, no. 1, pp. 47–67, 2019.

[24] D. Cabarcas, F. Göpfert, and P. Weiden, "Provably secure lwe encryption with smallish uniform noise and secret," in *Proceedings of the 2nd ACM workshop on ASIA public-key cryptography*, 2014, pp. 33–42.

[25] M. R. Albrecht, R. Player, and S. Scott, "On the concrete hardness of learning with errors," *Cryptology ePrint Archive*, 2015.

[26] V. Lyubashevsky, C. Peikert, and O. Regev, "On ideal lattices and learning with errors over rings," in *Annual international conference on the theory and applications of cryptographic techniques*, Springer, 2010, pp. 1–23.

[27] H. Chen, K. Lauter, and K. E. Stange, "Attacks on the search rlwe problem with small errors," *SIAM Journal on Applied Algebra and Geometry*, vol. 1, no. 1, pp. 665–682, 2017.

[28] M. Shao, Y. Liu, Y. Zhou, and Y. Shao, "On the security of lwe-based kems under various distributions: A case study of kyber," *Cryptology ePrint Archive*, 2024.

[29] J. Bos et al., "Crystals-kyber: A cca-secure module-lattice-based kem," in *2018 IEEE European symposium on security and privacy (EuroS&P)*, IEEE, 2018, pp. 353–367.

[30] J. Klemsa, "Hitchhiker's guide to the tfhe scheme," *Journal of Cryptographic Engineering*, 2023.

[31] C. Marcolla, V. Sucasas, M. Manzano, R. Bassoli, F. H. Fitzek, and N. Aaraj, "Survey on fully homomorphic encryption, theory, and applications," *Proceedings of the IEEE*, vol. 110, no. 10, pp. 1572–1609, 2022.

[32] Z. Brakerski and V. Vaikuntanathan, *Efficient fully homomorphic encryption from (standard) LWE*, Cryptology ePrint Archive, Paper 2011/344, 2011. [Online]. Available: `https://eprint.iacr.org/2011/344`.

[33] J. Fan and F. Vercauteren, "Somewhat practical fully homomorphic encryption," *Cryptology ePrint Archive*, 2012.

[34] C. Gentry, A. Sahai, and B. Waters, "Homomorphic encryption from learning with errors: Conceptually-simpler, asymptotically-faster, attribute-based," in *Annual cryptology conference*, Springer, 2013, pp. 75–92.

[35] L. Ducas and D. Micciancio, "Fhew: Bootstrapping homomorphic encryption in less than a second," in *Annual international conference on the theory and applications of cryptographic techniques*, Springer, 2015, pp. 617–640.

[36] M. S. Lee, "On the sparse subset sum problem from gentry-halevi's implementation of fully homomorphic encryption," *Cryptology ePrint Archive*, 2011.

[37] Z. Brakerski, "Fully homomorphic encryption without modulus switching from classical gapsvp," in *Annual cryptology conference*, Springer, 2012, pp. 868–886.

[38] Z. Brakerski and V. Vaikuntanathan, "Fully homomorphic encryption from ring-lwe and security for key dependent messages," in *Annual cryptology conference*, Springer, 2011, pp. 505–524.

[39] R. Agrawal and A. Joshi, "The ckks fhe scheme," in *On Architecting Fully Homomorphic Encryption-based Computing Systems*, Springer, 2023, pp. 19–48.

[40] J. H. Cheon, K. Han, A. Kim, M. Kim, and Y. Song, "Bootstrapping for approximate homomorphic encryption," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, Springer, 2018, pp. 360–384.

[41] A. Al Badawi and Y. Polyakov, "Demystifying bootstrapping in fully homomorphic encryption," *Cryptology eprint archive*, 2023.

[42] B. Li and D. Micciancio, "On the security of homomorphic encryption on approximate numbers," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*, Springer, 2021, pp. 648–677.

[43] B. Li, D. Micciancio, M. Schultz, and J. Sorrell, *Securing approximate homomorphic encryption using differential privacy*, Cryptology ePrint Archive, Paper 2022/816, 2022. [Online]. Available: `https://eprint.iacr.org/2022/816`.

[44] A. Al Badawi et al., "Openfhe: Open-source fully homomorphic encryption library," in *proceedings of the 10th workshop on encrypted computing & applied homomorphic cryptography*, 2022, pp. 53–63.

[45] H. Chen, K. Laine, and R. Player, "Simple encrypted arithmetic library-seal v2. 1," in *International conference on financial cryptography and data security*, Springer, 2017, pp. 3–18.

[46] Zama, *Concrete: TFHE Compiler that converts python programs into FHE equivalent*, `https://github.com/zama-ai/concrete`, 2022.

[47] S. Halevi and V. Shoup, "Design and implementation of helib: A homomorphic encryption library," *Cryptology ePrint Archive*, 2020.

[48] Z. Chen, *Seal-python*, `https://github.com/Huelse/SEAL-Python`, 2020.

[49] C. Dwork, "Differential privacy: A survey of results," in *International conference on theory and applications of models of computation*, Springer, 2008, pp. 1–19.

[50] J. Zhao, Y. Chen, and W. Zhang, "Differential privacy preservation in deep learning: Challenges, opportunities and solutions," *IEEE Access*, vol. 7, pp. 48 901–48 911, 2019.

[51] J. He and L. Cai, "Differential private noise adding mechanism: Basic conditions and its application," in *2017 American Control Conference (ACC)*, IEEE, 2017, pp. 1673–1678.

[52] C. Dwork, F. McSherry, K. Nissim, and A. Smith, "Calibrating noise to sensitivity in private data analysis," in *Theory of cryptography conference*, Springer, 2006, pp. 265–284.

[53] R. Sarathy and K. Muralidhar, "Evaluating laplace noise addition to satisfy differential privacy for numeric data," *Trans. Data Priv.*, vol. 4, no. 1, pp. 1–17, 2011.

[54] C. Dwork, "Differential privacy," in *International colloquium on automata, languages, and programming*, Springer, 2006, pp. 1–12.

[55] C. Dwork, A. Roth, et al., "The algorithmic foundations of differential privacy," *Foundations and trendső in theoretical computer science*, vol. 9, no. 3–4, pp. 211–407, 2014.

[56] S. Kotz, T. Kozubowski, and K. Podgorski, *The Laplace distribution and generalizations: a revisit with applications to communications, economics, engineering, and finance*. Springer Science & Business Media, 2012.

[57] Q. Ke and T. Kanade, "Robust l/sub 1/norm factorization in the presence of outliers and missing data by alternative convex programming," in *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, IEEE, vol. 1, 2005, pp. 739–746.

[58] P. Kairouz, S. Oh, and P. Viswanath, "The composition theorem for differential privacy," in *International conference on machine learning*, PMLR, 2015, pp. 1376–1385.

[59] F. D. McSherry, "Privacy integrated queries: An extensible platform for privacy-preserving data analysis," in *Proceedings of the 2009 ACM SIGMOD International Conference on Management of data*, 2009, pp. 19–30.

[60] J. Lee and C. Clifton, "How much is enough? choosing ε for differential privacy," in *International Conference on Information Security*, Springer, 2011, pp. 325–340.

[61] Synthea, *Synthea patient generator*, `https://github.com/synthetichealth/synthea`, 2016.

[62] N. Holohan, S. Braghin, P. Mac Aonghusa, and K. Levacher, "Diffprivlib: The IBM differential privacy library," *ArXiv e-prints*, vol. 1907.02444 [cs.CR], Jul. 2019.

[63] A. Aktay et al., "Google covid-19 community mobility reports: Anonymization process description (version 1.1)," *arXiv preprint arXiv:2004.04145*, 2020.

[64] C. T. Kenny, S. Kuriwaki, C. McCartan, E. T. Rosenman, T. Simko, and K. Imai, "The use of differential privacy for census data and its impact on redistricting: The case of the 2020 us census," *Science advances*, vol. 7, no. 41, eabk3283, 2021.

[65] L. Bergerat et al., "Parameter optimization and larger precision for (t) fhe," *Journal of Cryptology*, vol. 36, no. 3, p. 28, 2023.

[66] A. Viand, C. Knabenhans, and A. Hithnawi, "Verifiable fully homomorphic encryption," *arXiv preprint arXiv:2301.07041*, 2023.

[67] B. Ghaleb and W. J. Buchanan, "Side channel analysis in homomorphic encryption," *arXiv preprint arXiv:2505.11058*, 2025.

[68] F. Aydin, E. Karabulut, S. Potluri, E. Alkim, and A. Aysu, "Reveal: Single-trace side-channel leakage of the seal homomorphic encryption library," in *2022 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, IEEE, 2022, pp. 1527–1532.

[69] C. Krüger, B. Moriya, and D. Schoop, "A performance comparison of the homomorphic encryption schemes ckks and tfhe," *Cryptology ePrint Archive*, 2025.

[70] I. G. Cohen and M. M. Mello, "Hipaa and protecting health information in the 21st century," *JAMA*, vol. 320, no. 3, pp. 231–232, Jul. 2018, ISSN: 0098-7484. DOI: 10.1001/jama.2018.5630.

[71] C. J. Hoofnagle, B. Van Der Sloot, and F. Z. Borgesius, "The european union general data protection regulation: What it is and what it means," *Information & Communications Technology Law*, vol. 28, no. 1, pp. 65–98, 2019.