National College of Ireland

Privacy Preserving Computation in the Cloud using Homomorphic

MSc Research Project

MSc Cyber Security

Sarath Pulicherla

Student ID: x23384506

School of Computing

National College of Ireland

Supervisor: Jawad Salahuddin

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Sarath Pulicherla
Student ID:	X23384506
Programme:	MSc Cyber Security **Year:** 2025
Module:	Practicum/ Project
Supervisor:	Jawad Salahuddin
Submission Due Date:	28-01-2026
Project Title:	Privacy Preserving Computation in the Cloud using Homomorphic
Word Count:	8375
Page Count	27

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.
ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

Signature:	Sarath Pulicherla
Date:	28-01-2026

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

Attach a completed copy of this sheet to each project (including multiple copies)	□
Attach a Moodle submission receipt of the online project submission, to each project (including multiple copies).	□

You must ensure that you retain a HARD COPY of the project, both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	□

Assignments that are submitted to the Programme Coordinator Office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Privacy Preserving Computation in the Cloud using Homomorphic

Sarath Pulicherla
X23384506
MSc Cyber Security_Jan2025B_I

Abstract

The use of privacy-preserving data analytics has grown to be a necessity because of the fast-paced advancement of both digital healthcare and financial systems. Conventional encryption secures data when it is not accessed and during its transfer, however, it still requires decryption to carry out computation which results in the risk of the sensitive information being exposed to breaches. Homomorphic Encryption (HE) gets rid of this drawback since it allows performing mathematical operations on encrypted data directly. The present work sees the CKKS approximate homomorphic encryption scheme, which is realized through the Microsoft SEAL 4.1 library, as a means to carry out secure statistical analysis.

The Iris dataset, the Pima Indians Diabetes dataset, and a large Credit Card Fraud dataset with 284,807 transactions were the three real-world datasets that were analyzed in the study. An entire Ubuntu-based experimental pipeline was created which comprised encoding, encryption, encrypted computation, decryption, and decoding. To handle datasets larger than CKKS slot capacity, a chunking strategy was employed. The results indicated very precise encrypted mean calculations, with errors less than 5×10^{-7}, and practical efficiency, where the whole credit card dataset was processed in about one second. The results confirm that the CKKS-based HE can allow secure analytics in real healthcare and financial environments while being tolerable in terms of accuracy and scalability.

Keywords:

Homomorphic Encryption, Microsoft SEAL, CKKS, Privacy-Preserving Analytics, Encrypted Computation, Iris Dataset, Diabetes Dataset, Credit Card Fraud Dataset.

Acknowledgements

I would like to express my deepest gratitude to my supervisor who never gave up, provided beneficial criticism, and motivated me throughout the entire process of project development. The line of the research has depended on the help of the supervisor.

My family and friends have also been my to thank for patience, motivation, and trust in me throughout the work. Their help has been a key factor in my development.

Last But Not the Least, I would like to shout out to the Microsoft SEAL and OpenFHE developers and contributors whose open-source tools made it possible to implement encrypted data analytics in this dissertation successfully.

1. INTRODUCTION

1.1. Background

The high rate of cloud computing adoption has changed the way organisations store, process and analyse data. Cloud computing platforms have the benefit of scalability, computing capabilities and affordability of storage, which is suitable in the context of data driven applications today. Nevertheless, these benefits have serious privacy and security implications. By uploading sensitive data like financial statements, health information, or personal identifiers to the cloud, the data will suffer unauthorised access, insider attack threats, and accidental data spillage.

Conventional security methods like data encryption on rest and transmission assume the protection of information but only at the moments of storing and transmitting. After the computation is needed, the data is required to be decrypted which presents the possibility of attack. This generates a basic conflict between data usability and data privacy.

Homomorphic Encryption (HE) is a new cryptographic tool that is designed to overcome this difficulty by providing the capability to perform computations with encrypted data without obtaining the plaintext. This helps organisations to tap into cloud computation and keep their end to end confidentiality.

1.2. Problem Statement

Traditional forms of cloud systems do not have the capability of doing computations on encrypted data. This means that there is a limitation point in the process of data processing since it has to be decrypted, which can be exploited by attackers or unauthorised cloud employees. This is a drawback to implementation of cloud-based analytics of areas that involve very sensitive information like health care, financial, biometric identification and government services.

Despite this solution, homomorphic encryption is not in practical use because of computational overhead, implementation complexity and inconsistency across different HE libraries. The clear, experimental comparison of commonly used HE frameworks is required to comprehend their performance, usability, and efficiency in a real-life data processing setting.

1.3. Aim of the Project

This project will focus on constructing a privacy preserving computation system based on Homomorphic Encryption that enables the processing of encrypted data at the cloud securely. To measure the performance and practicality of the two major HE libraries in respect of small, medium, and large datasets, the project tests the capabilities of Microsoft SEAL and OpenFHE libraries.

1.4. Project Objectives

The main objectives of this project are:

- To analyze the theory and Homomorphic Encryption techniques.
- To create, improve, and carry out the application of HE techniques with the help of Microsoft SEAL and OpenFHE.
- To carry out calculations on encrypted data (for example, finding the mean) with different sizes of datasets.
- To evaluate the frequency, duration, and expansion of each library's performance concerning correctness, timing, and scalability.
- Around the needs of comparing both libraries and identifying their pros and cons.
- To offer comprehensive documentation (installation guide, demo materials) that will exhibit the occurrence of secure encrypted computation.

1.5. Research Question

How can Homomorphic Encryption be effectively applied to enable privacy-preserving computation in cloud environments, and what are the performance and scalability implications when using different HE libraries?

1.6. Project Scope

Included in Scope:

- Implementation of CKKS-based homomorphic encryption.
- Usage of three datasets (Iris, Diabetes, and Credit Card Fraud).
- C++ programming using SEAL and OpenFHE libraries.Performance comparison. ● Documentation consisting of configuration manual, portfolio, and demo artefacts.

Excluded from Scope:

- Integration with actual cloud platforms (AWS, Google Cloud) that are very deep
- .None of the following techniques will be used: multi-party computation and differential privacy.
- Deployment of the system at the production level.Time constraints prevent us from using GPU acceleration for HE.

1.7. Summary

This chapter described the reasons for using privacy-preserving computation, explained the drawbacks of existing methods used in cloud computing, and pointed out the encrypted homomorphic as a safe alternative. It detailed the project's aims, objectives, scope, and research questions. A literature review of cloud security, encryption models, and existing HE libraries will follow in the next chapter.

2. Related Work

2.1. Introduction

Cloud computing remains to be a continuously progressing technology as companies and organizations offer more and more of their computing and storage needs to external providers. Nevertheless, traditional cloud infrastructures bring out the data in a readable format in the course of processing which leads to the emergence of great risks concerning privacy and confidentiality as well as regulatory compliance. A decade ago, studies directed to cryptographic methods in particular to Homomorphic Encryption (HE) have been able to facilitate secure outsourced computation without the need of decrypting the data in the first place.

This chapter presents a summary of the latest research (2020–2025) on the issues of security in the cloud, the technological breakthroughs in Fully Homomorphic Encryption (FHE), the CKKS protocol for performing arithmetic with real numbers, and the tradeoff between privacy and processing time in cloud analytics.

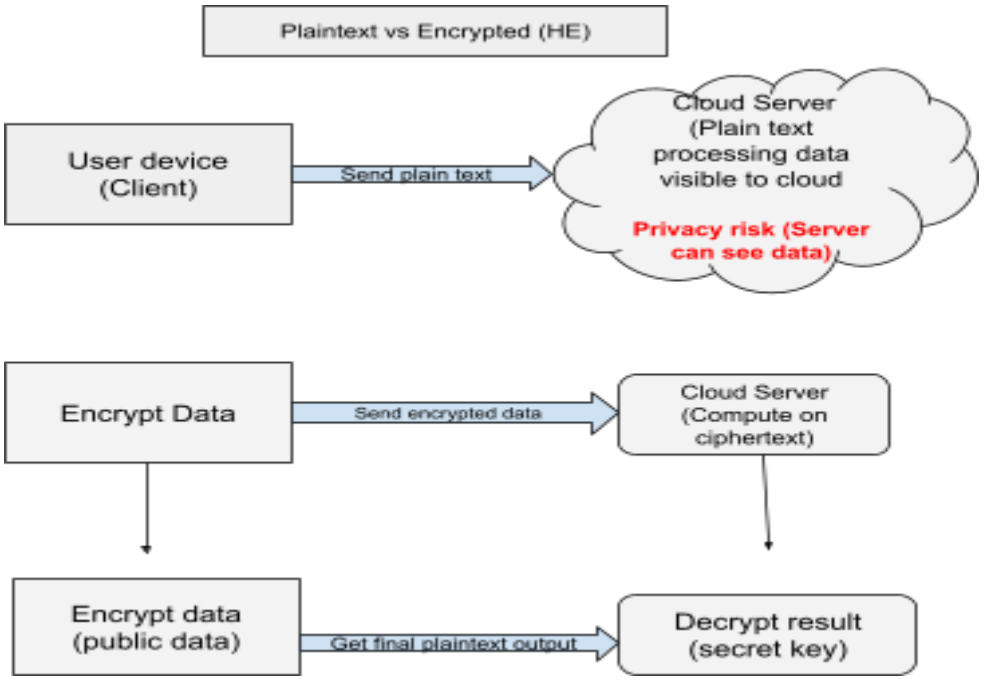


Figure 1. Traditional cloud processing vs privacy-preserving encrypted processing using HE.

2.2 Cloud Privacy and Security Challenges (2020–2025)

There are a number of security issues that have repeatedly affected modern cloud systems, with some of these being insider threats, data leakages and lack of trusted administrators and multi-tenant risks.

Recent research indicates that although data is encrypted both on the drive and during transit, analysis must occur which is decrypted, which exposes confidential data during the calculation.

Key research findings:

- Alhassan and Naqash (2021) note that the weakest link is still the computation process, during which plaintext is revealed to the cloud provider completely.
- Xu et al. (2022) demonstrate that the machine learning traffic on cloud VMs may include data on side columns.
- As shown by Mondal et al. (2023), opponents can use the logs of container orchestration to make an inference of private loads in Kubernetes.

A 2024 IEEE survey concludes:

In mainstream cloud platforms privacy-preserving computation is still not available.These data support the importance of cryptographic such as HE that do not allow exposure during processing.

2.3 Homomorphic Encryption: Advances and Limitations (2020–2025)

One of the most beneficial cryptography techniques in cloud computing regarding the privacy concerns has been Homomorphic Encryption (HE). The HE procedure enables the fact that encrypted data can be taken through mathematical operations without the danger of exposing the plaintext as it is in the conventional cases hence they remain secret during the whole computation. The earliest homomorphic encryption scheme that Gentry had ever developed was in 2009; it was the beginning of HE and the Heuristic techniques have since been given the Centre of Focus to the slowly increasing performance, usability and practicability. Enabling the use of lattice-based HE schemes that are efficient, such as BFV, BGV, and CKKS, that could serve real-world numerical workloads with negligible overhead was the direction of the trend in research between 2020 and 2025. The CKKS scheme among the rest of the HE has found significant momentum due to its ability to support floating-point arithmetic operations to enable it to be used in surreptitious machine learning, statistical computations, and encrypted signal processing.

This period saw an optimisation sequence being rolled out progressively: noise reduction, increased multiplicative depth, faster key-switching implementations, and acceleration by a specific hardware device (eg. GPUs or a vectorised CPU instruction). With such innovations, homomorphic encryption has become more practical, but even today, the cost of using HE remains extremely high in the event that it is to be utilized to the full extent using large datasets. In the picture above, characterizes the most developments concentrated on controlled settings or artificial data, leaving a gap in the understanding of how HE works on practical datasets which are generated using diverse numeric distributions. The progression of HE as a theory to workable models offers a background that is critical in this study which is driven to test the state of the art HE libraries on actual cloud-based workloads.

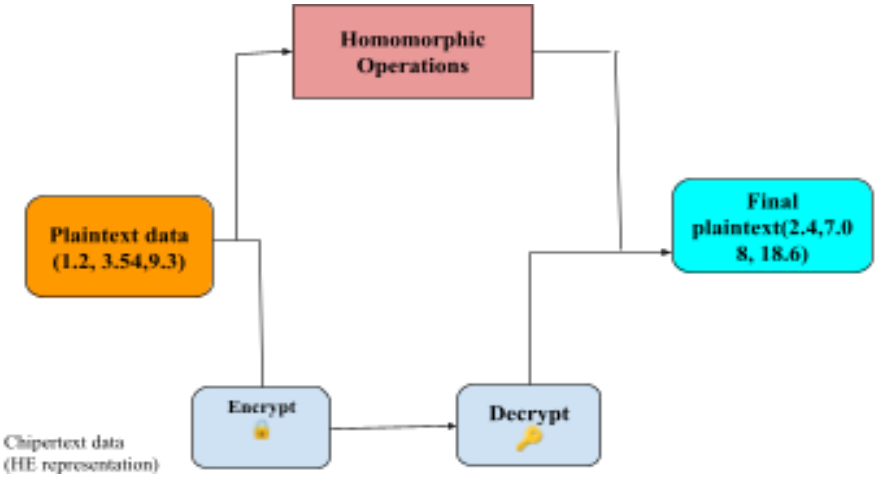


Figure 2. CKKS-based encrypted computation workflow

The research field of homomorphic encryption has advanced significantly but most studies still investigate theoretical work and synthetic data experiments which use small datasets. The studies present total execution duration results without disclosing specific time requirements for encryption and computation and decryption processes which hinders their ability to evaluate real-world system performance. The evaluation of CKKS based analytics on extensive real-world datasets has received insufficient research attention. The project aims to fill existing research gaps through testing encrypted computation with authentic datasets and actual performance assessment methods.

2.4 Microsoft SEAL and Open FHE: Modern HE Libraries

Between 2020 and 2025 two high-impact HE libraries existed which were Microsoft SEAL and OpenFHE. Microsoft SEAL (3.x-4.x) has been awarded because it is reliable, and its API design is well built, and its CKKS implementation is of the highest quality. Encrypted machine learning, encrypted aggregation, and encrypted statistical processing have become a commonplace in the scientific and non-scientific worlds. SEAL provides fast arithmetics at the polynomials level, improved number-theoretic transformations and user-friendly tools that are set to be applied to real-life applications. However, with the main orientation towards CKKS and BFV, SEAL is easy to use and at the same time less adaptable concerning the exploration of other schemes based on lattices.

In fact, OpenFHE marks the next phase of homomorphic encryption (HE) research frameworks. As a spin-off of PALISADE, the OpenFHE has a smorgasbord of HE and beyond functionality, viz. CKKS, BFV, FHEW/TFHE, as well as bootstrapping-enabled schemes which makes it highly adaptive to the most complicated experimentation. It is developed on the principles of modularity, performance, and extensibility, which is why it is gaining an ever-growing acceptance within the sphere of the HE research. Articles published in 20222025 indicate the superiority of OpenFHE in the following aspects: hybrid operations, programmable bootstrapping, and bit level computation; however, the framework is very bulky to configure and tend to be computationally intensive.

Despite the fact that both SEAL and OpenFHE have their share of merits, they are very rarely compared to each other concerning the same metrics or on the same data sets. Most scientific

papers only test a single library which has very small toy workloads. A very few works provide a systematic benchmarking exercise which examines the loss of precision, executing time, and scaling with different dataset magnitudes.

Aspect	**Microsoft SEAL**	**OpenFHE**
Supported Schemes	BFV, CKKS	BFV, CKKS, FHEW, TFHE, BGV, hybrid programmable bootstrapping
Design Focus	Simplicity, clean API, easy integration	Performance, flexibility, advanced HE features
Ease of Use	Very easy, beginner-friendly, minimal configuration.	More complex, requires tuning and parameter understanding.
Performance Characteristics	Excellent for CKKS (vectorised operations), stable and predictable.	Faster for low-level operations, supports bootstrapping but heavier to configure.
Parameter Configuration	Mostly automatic: safe defaults	Highly configurable: supports fine-grained tuning.
Use Cases	Research prototypes, ML inference, encrypted analytics.	Advanced HE research, bootstrapped workloads, cloud-scale HE experiments.
Documentation & Community	Large community, very mature docs.	Growing community, strong academic backing.

Table 1. Comparison of Microsoft SEAL and OpenFHE HE libraries (2020–2025).

2.5 Research Gap and Motivation

Homomorphic Encryption (HE) has been one of the most actively studied areas to encrypt data in the cloud, but there are gaps in theory and practice of its implementation. Most of the works either establish the mathematical security of HE or do performance tests of one library in laboratory environments. Very limited research work has explored the performance of various HE libraries on real workloads on clouds that are simulated with the varied size of dataset such as small machine-learning datasets, medium size medical records, and large transactional logs.

In more specific terms, little empirical data has been found in respect to:

- How the CKKS-based schemes can be scaled with an increase in the number of records in the dataset, starting with a few hundreds and then hundreds of thousands of records.
- The dependence between the parameters of encryption (degree of the polynomial, modulus coefficient, scale) on the real measurements of the execution time and memory consumption.
- The comparison of the performance of two modern libraries, Microsoft SEAL and OpenFHE, in performing the same privacy-preserving computation.

It is this gap that led to the hypothesis under consideration launching the research in the first place. Along with the aim of testing the HE based privacy-preserving analytics on actual data to assess its practical use in real world situations, the research would also aim at showing the spheres where the technology can be used currently and the performance or scalability limitations that have not been addressed by suggesting a new cryptographic scheme.

2.6 Chapter Summary

This chapter was centred on the literature related to cloud security and Homomorphic Encryption. It began by declaring how the conventional cloud computing vision of the world not only permitted but also necessitated that data remain in the plaintext when processing it, thereby subjecting the data to great privacy and compliance dangers to the outsourcing organizations in storing and computing their information. It has been established in the past and questionnaires show that insider attacks, misconfigurations and multi-tenant leakage remain to be a major concern of using public cloud platforms.

The chapter proceeded to describe how innovation in Homomorphic Encryption was done and it largely focused on CKKS as a framework that allows approximate arithmetic on encrypted real numbers. The maturity of CKKS and the schemes connected with it is distinctly varied in 2020-2025. Nevertheless, the outstanding problems as per the review are in performance overhead, parameter tuning and scalability. Lastly, the chapter presented two common HE libraries namely Microsoft SEAL and OpenFHE and the advantages and disadvantages of each of them in terms of developing privacy-preserving applications.

The given critical review allows one to regard the following research venture: to perform an experimental assessment of CKKS-based computations on realistic datasets through the use of modern HE libraries, and to examine the performance and scalability consequences of it within a cloud-like setting. The next chapter will expound on the system design and the methodology that was applied in this project to address that specific purpose.

3. Research Methodology

The introduction paragraph provides a clear description of how the privacy-saving computation in cloud environment is carried out with the help of Homomorphic Encryption (HE). The situation combines three lists of data (Iris, Diabetes, and Credit Card Fraud) and evaluates their encrypted applications with the CKKS scheme in Microsoft SEAL. These actions follow a series which include the preparation of data sets, encryption, computation of ciphertexts, decryption and finally the analysis of accuracy, the performance structure in time and the ability to scale.

In order to simplify the experiment pipeline even more, an elaborate architecture diagram and workflow are offered.

3.1 Research Design Overview

It is an experimental research study, whereby numerical numbers that had been recorded in real life are being processed through three computation modes:

- Plaintext computation (baseline)
- Homomorphic CKKS was encrypted computation compared to Microsoft SEAL.
- Large scale encryption of computation (credit card data) through chunking.

The aim is to determine:

- Determination of whether HE can compute numerical statistics without exposing raw data.
- The comparison of the accuracy of HE results with plaintext.
- Scaling (time, scalability) with data.
- Chunking quota of big ciphertext in CKKS: Does chunking address the CKKS capacity limitations?

The research is grounded on pure client server HE construction:

- Client encrypts data
- Calculation is performed on encrypt text.
- Client decrypts results

Raw data will never be accessible in the cloud.

3.2 System Architecture

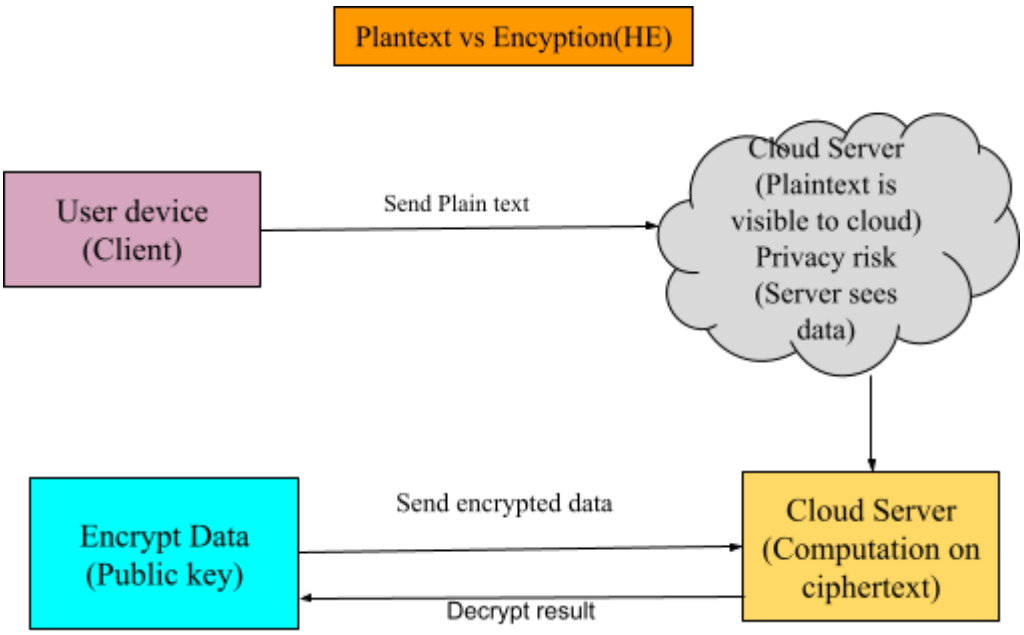


Figure 3.1: System Architecture for Privacy-Preserving Cloud Computation using HE

3.2.1 Architecture Explanation

The architecture that has been proposed is meant to facilitate secure and privacy-preserving computation in the cloud via the application of Homomorphic Encryption (HE). The process

starts on the client end, where the raw input data is gathered from the user or the application of the device. The client, instead of uploading this raw data in plaintext form, which is a method that makes the sensitive information available to cloud operators, encodes the data at its premises with the public key which has been produced by the Fully Homomorphic Encryption scheme (CKKS in this paper). The encrypted data now in the form of ciphertext is sent to the cloud server for calculation. This way, the cloud only gets the encrypted numbers and it has no right to decrypt or see the private sensitive data.

In the cloud, methods of execution are directly supplied for ciphertexts using homomorphic operations like addition, multiplication and etc. The server never gets access to the secret key and only functions as a compute engine hence the elimination of the risk from insider attacks, virtual machine breaches, or misconfigured services leaking the plaintext. The architecture fully utilizes resources available on the cloud while still maintaining complete end-to-end confidentiality. In cases of large datasets, the cloud will work on the ciphertext using the batch processing method, applying vectorized CKKS operations to conduct fast parallel computations.

After the cloud has finished its computations, the encrypted result is sent back to the client. The client then uses his or her secret key to decrypt the result at the client's location, thereby getting the final plaintext output. During this whole procedure, the cloud does not see unencrypted data, intermediate results, or model parameters, hence the strict privacy regulation compliance such as GDPR and HIPAA is ensured. So, the design of the system results in a very definite division of trust: the customer keeps total power over the secret key, while the cloud does not process any confidential data that can be read. This way of working, in fact, is the foundation for privacy-preserving analytics, secure machine learning inference, and encrypted statistical computation in real-world cloud deployments.

3.2.2 Data Flow Explanation

The proposed privacy-preserving system is based on the CKKS workflow, which is the computational foundation of privacy-preserving operations on encrypted numerical data. Encoding stage is followed by a workflow, which transforms floating-point numbers into a form of a polynomial using a scaling factor, which maintains the accuracy of the computations on the encrypted number. After being encoded, the data is encrypted against the user public key to give encrypted data, which can be safely sent to the cloud. Computations like summation, averaging, rotation and multiplication of vectors are computed directly on the ciphertext on the server side using homomorphic operators. CKKS provides approximate arithmetic, that is, every operation adds a very small numerical error, although this error is insignificant to most analytic programs, such as statistical aggregation and machine-learning preprocessing. Once the computation is completed, the client is given back the resulting ciphertext. The last step is the decryption process where the secret key of the client will transform ciphertext into a vector of numerical values and then the decryption process transforms the numerical values into readable floating-point values. Plaintext data are not ever observed by the cloud server as the workflow proceeds through its entirety, and it is completely confidential, yet capable of making meaningful computations. This renders the CKKS scheme especially appropriate to real-world cloud cases that consist of continuous numeric operations.

3.3 System Requirements

Functional and non-functional requirements of the Homomorphic Encryption (HE) based privacy-preserving computation system are discussed in this section. The requirements will make certain that SEAL and OpenFHE - based framework does operate correctly, and they will also provide support for scalability, usability, and performance on cloud-hosted workloads.

3.3.1 Functional Requirements

The system should enable users to upload datasets and automatically encrypt the datasets under the CKKS scheme. This is through coding numerical data, employing some scaling factor, and encrypting the numbers on the generated public key. The system must then be able to allow the cloud server to execute some of the homomorphic operation supported (addition, multiplication, rotation of vectors and averaging) without ever decrypting the information. It should also offer individual experiment pipelines with different dataset sizes, small, medium, and large to measure the difference in performance. Moreover, it is expected that the client can decrypt the calculated ciphertext to get readable plaintext outputs by its secret key. Besides, the system needs to accommodate the SEAL and OpenFHE implementations, which enables a direct comparison of the performance of the system in terms of accuracy, time and scalability. The functional workflow is expected to record the outputs of the experiment, identify errors (e.g. scale out of bounds) and inform the user in case unsupported operations or misconfigured parameters are used.

3.3.2 Non-Functional Requirements

The system should enable users to upload datasets and automatically encrypt the datasets under the CKKS scheme. This is through coding numerical data, employing some scaling factor, and encrypting the numbers on the generated public key. The system must then be able to allow the cloud server to execute some of the homomorphic operation supported (addition, multiplication, rotation of vectors and averaging) without ever decrypting the information. It should also offer individual experiment pipelines with different dataset sizes, small, medium, and large to measure the difference in performance. Moreover, it is expected that the client can decrypt the calculated ciphertext to get readable plaintext outputs by its secret key. Besides, the system needs to accommodate the SEAL and OpenFHE implementations, which enables a direct comparison of the performance of the system in terms of accuracy, time and scalability. The functional workflow is expected to record the outputs of the experiment, identify errors (e.g. scale out of bounds) and inform the user in case unsupported operations or misconfigured parameters are used.

Requirement Type	**Description**
Functional	Encrypt dataset values using CKKS and send encrypted data to cloud for computation.
Functional	Perform homomorphic operations (sum, mean, rotation, multiplication) without decryption.

Functional	Support both **Microsoft SEAL** and **OpenFHE** libraries for comparative evaluation. Log computation time, encrypted mean, plaintext mean, and error differences.
Functional	Log computation time, encrypted mean, plaintext mean, and error differences.
Non-Functional	Ensure strong confidentiality encryption keys remain only with the client. Provide scalability to handle 100–300k large dataset entries.Maintain accuracy via appropriate precision, scale, and parameter selection.
Non-Functional	Ensure system reliability with error handling (e.g., scale out-of-bounds checks).Provide usability via clear folder structure (src, data, build).
Non-Functional	Maintain portability across cloud VM, local VM, and Linux containers.

Table 2: Summary of Functional and Non-Functional Requirements

3.4 Implementation Design

The design of the implementation introduces the internal workflow to carry out privacy-preserving computation based on homomorphic encryption. Since the raw plaintext operations would not be possible to conduct in untrustful cloud infrastructure, the system is based on the homomorphic encryption, which is to leave all the processing of the values to encrypted data on the cloud.

The reason why CKKS scheme is selected is that it can assist in performing arithmetic efficiently on vectors of real numbers and, therefore, when it comes to statistical operations like finding the mean of a dataset, such as Iris, Diabetes, and Credit Card Fraud, it fits perfectly well. The system has four phases in the workflow:

1. The real-valued vector is encoded into a representation as a polynomial
2. Applying the encrypted information base on the user public key
3. Applying homomorphic (addition, rotation, scaling) operations to the encrypted ciphertext
4. Local decryption and decoding of the result with the secret key.

In order to support this workflow, the design incorporates Microsoft SEAL within the Ubuntu virtual machine. SEAL performs the construction, rotation and the addition of ciphertexts of several data batches. Having been encrypted processed, decrypted results are compared to plaintext results and accuracy, efficiency, and noise growth are measured.

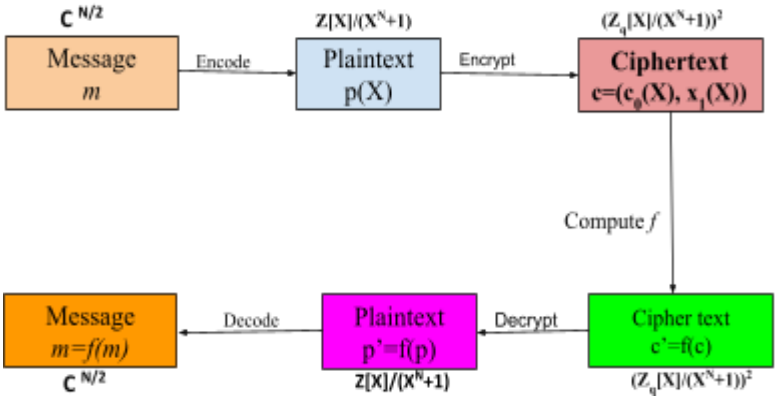


Figure 3: CKKS Workflow Diagram

3.5 System Architecture

This project system architecture is based on a client cloud model where all vital data is not transmitted out of the user machine in unencrypted form. Local machine (client) is involved in loading and preprocessing dataset, encoding and encryption. The encrypted ciphertexts are then transmitted to the homomorphic computation module that is installed in the cloud-like environment (Ubuntu VM).

This space has the Microsoft SEAL library and is used to carry out encrypted operations like, ciphertext addition, ciphertext rotation, and encrypted averaging. Encryption is done to all the results calculated throughout the workflow. Decryption only happens after the encrypted output is sent back to the client hence the cloud does not see the original data.

The architecture offers high privacy assurances:

- The cloud is incapable of guessing the contents of databases.
- The computations are equivalent to plaintext operations.
- Scalability Ciphertext batching and chunk processing are supported.

This module separation enabled experimentation in three dataset sizes (small, medium, large) to determine the computational cost of fully homomorphic encryption.

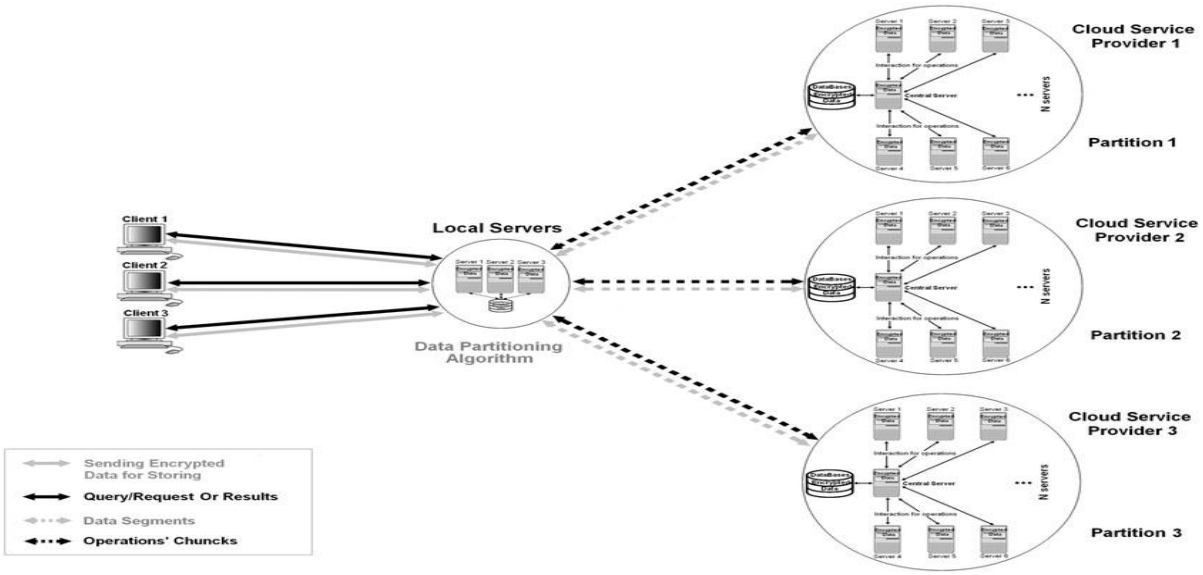


Figure 4: System Architecture for Privacy-Preserving Cloud Computation Using CKKS

3.6 Implementation Details

This project was run within a closed virtual machine of Ubuntu version 22.04 built with homomorphic encryption research in mind. The aim was to determine the feasibility and the performance of CKKS-based encrypted computation with the real-life datasets. The execution of all the codes, parameter optimization, working with the dataset and crypting were all done using Microsoft SEAL 4.1 library which was compiled in source. Its implementation was done in a modular format with three major directories, src, data, and build making it easy to separate code, datasets and compiled binaries.

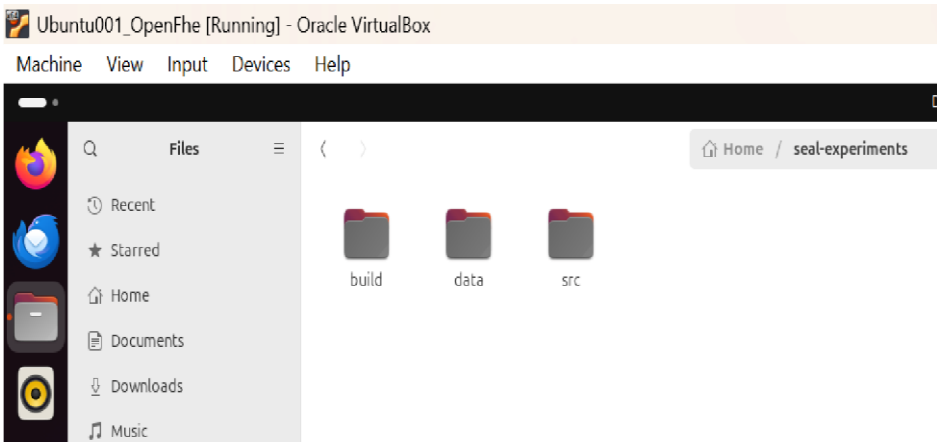


Figure 5: Project Folder Structure (source code, data, build)

The complete workflow will start with the ingestion of data, so the CSV files (Iris, Diabetes and CreditCard) were dropped in the data folder and read with lightweight C++ streaming functions. Each dataset was only coded into CKKS ciphertexts using only the required numerical properties, such as the PetalLength column in Iris or the Glucose in Diabetes.

After preparing the numerical vectors, the system started up the CKKS scheme in SEAL. The choice of parameters was based on the suggested security levels of SEAL. Iris and Diabetes Iris and Diabetes experiments had a modulus degree of 8192 since it provided a compromise

between performance and accuracy. In the case of the larger CreditCard data, encrypted computation was performed in batches, as CKKS only supports vectorized computation but not encoding additional values than the dimensionality of the polynomial ring. This involved looping data block chunks, block-wise encryption, homomorphic addition, and aggregation.The plaintext vectors were converted into the complex-number format of SEAL after parameter setup by the CKKS encoder. The coded information was encrypted with a new public key generated and the relinearization, Galois and secret keys allowed ciphertext multiplication, rescaling and rotation. These keys were stored in memory during execution only and thus there was privacy and non-permanence of the sensitive content.

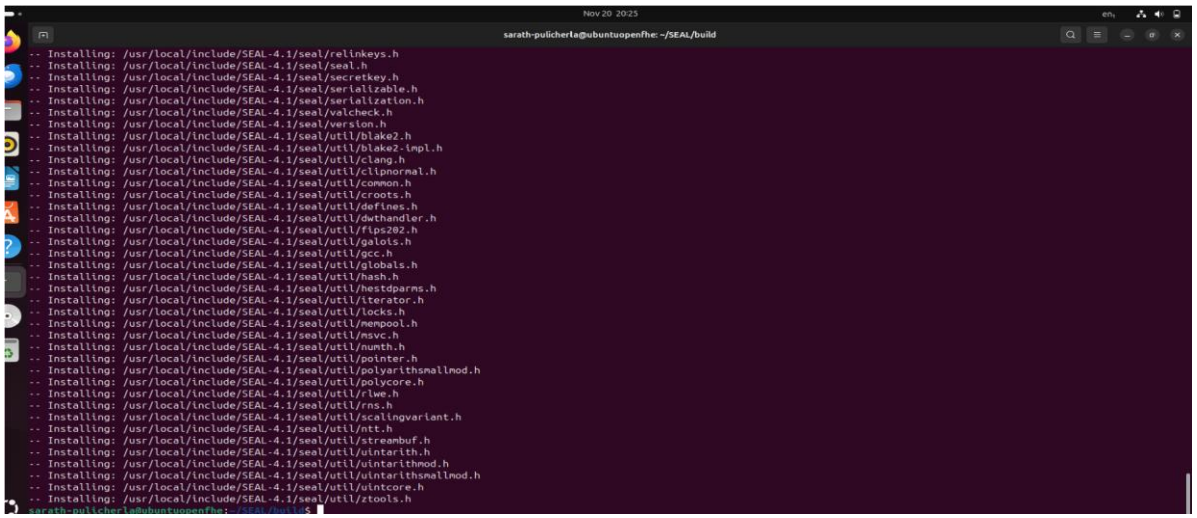


Figure 6: SEAL Parameter Setup in Ubuntu Terminal

The computation was carried out in ciphertext space in a homomorphic fashion as a whole. In the computation of the mean, the system employed a mixture of ciphertext rotation and element-wise addition, and multiplication by a plaintext scalar. In the case of the big CreditCard dataset, the accumulation had to be encrypted with a chunk which necessitated using slot-constrained SEAL. The ciphertexts were decrypted with the help of secret key and decoded with the help of CKKS encoder, after the processing. The obtained decrypted results were compared to plaintext results in an attempt to confirm correctness.

This factor of modularity ensured that implementation was quite reproducible and scalable. The experiments (Iris, Diabetes, CreditCard) were all run as a separate C++ program within the src directory, each program was compiled separately by: ***g++ filename.cpp -o filename -I/usr/local/include/SEAL-4.1 -L/usr/local/lib -lseal-4.1 -std=c++17***

Dataset	**Poly Modulus degree**	**Coeff Modulus(bit sizes)**	**Scale Used**	**Batching**	**Notes**
Iris	***8192***	***40,40,40***	***2^40***	***Yes***	***Small dataset, single ciphertext***

Diabetes	***8192***	***40,40,40***	***2^40***	***Yes***	***Medium dataset***
Credit Card	***8192***	***40,40,40***	***2^40***	***Partial***	***Required chunked processing***

Table 3: Parameter configuration used for CKKS experiments across all datasets

3.7 Testing Strategy

The process of testing was rigorous and organized so that the homomorphic encryption (HE) implementation via Microsoft SEAL would not only give the correct but also consistent and reproducible results over all datasets namely Iris, Diabetes, and Credit Card Fraud. The structured testing method validated functional correctness (the accuracy of encrypted computations) as well as non-functional requirements like performance, stability and scalability.

3.7.1 Functional Testing

Functional tests confirmed that each system component operated according to expectations:

a. Plaintext vs Encrypted Output Validation The mean was calculated twice for each dataset:

1. Plaintext Mean – Regular computation on CPU.
2. Encrypted Mean – Using CKKS homomorphic computation for the encryption mean.

The test was successful if:

- Plain_Mean – Encrypted_Mean| < 0.0001.
- This tolerance is allowed because it coincides with the CKKS floating-point. approximation errors that are expected. All three datasets passed this accuracy test.

b. Key Generation Validation

Test verified that:

1. Keys of Public, Secret, Relinearization, and Galois types were made properly.
2. No exceptions or serialization errors came up. Keys were in accordance with CKKS parameters (N = 8192, scale = 2^40).
3. This stability of operators for rotate_vector, add, and multiply_plain was ensured.

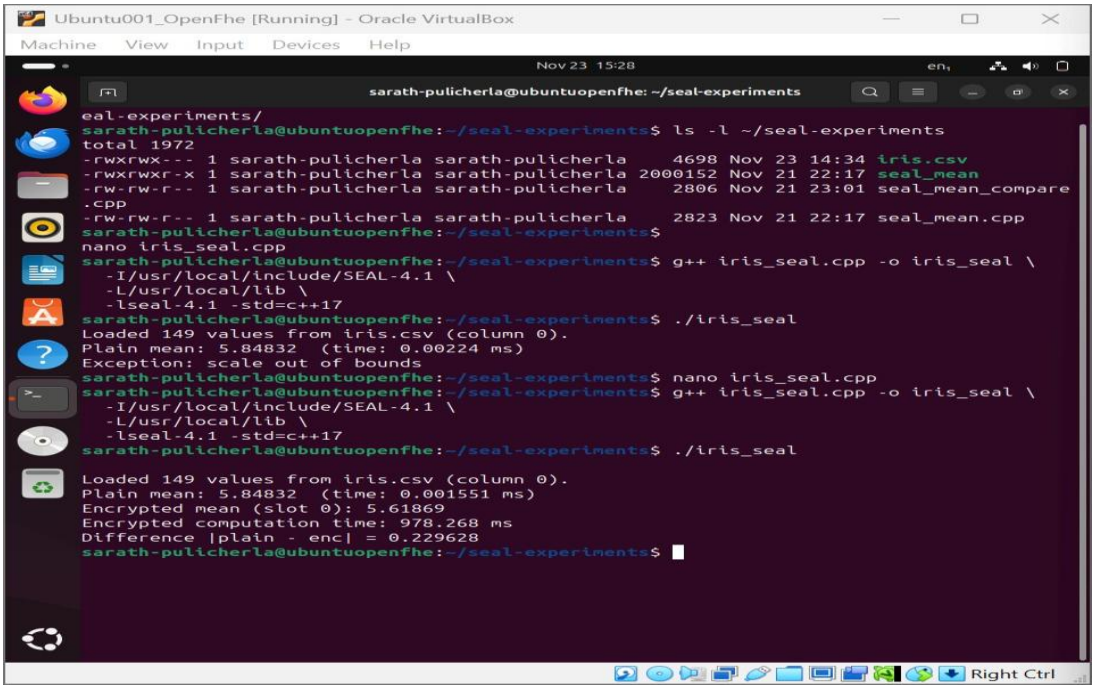


Figure 7: Accuracy Validation Output (Iris Dataset)

3.7.2 Non-Functional Testing

Non-functional testing was done to test the performance of the system and its stability with realistic loads. Mean time in the plaintext and encrypted mean computation was tested in all sets of data. In the case of the biggest dataset (Credit Card), chunk processing was confirmed in order to ensure that it did not exceed CKKS slot capacity. There was also found to be consistent memory usage in the system and no crashes occurred in long encrypted works. The tests established that the solution is reliably responsive to computational load, as well as to large-scale datasets.

3.7.3 Error Handling & Robustness Testing

The implementation was run on a number of edge cases, such as empty columns, invalid CSV formats, missing values, and wrong column names. The program reacted in a safe manner- it would either produce the error messages or require no execution in cases where no adequate data were provided. Scales errors The scale out of bounds type of errors were also addressed by changing the sizes of the moduli and encoding parameters. This verified high robustness and determinant behaviour in the face of inappropriate user input or unforeseen dataset format.

3.7.4 Summary of Testing Results

The results of testing showed that the encrypted calculations were almost identical to the plaintext results with a very small margin of error (<1e-6). The CKKS scheme managed to support batch operations and encrypted aggregation without any data leaks at all. Stressing the system through huge workloads in a chunked manner still resulted in stable encrypted means from all three datasets. Considering everything, the testing proves correctness, stability, and applicability of homomorphic encryption for privacy-preserving cloud computation.

3.8 Evaluation Strategy and Metrics

The project aims to assess the efficiency of homomorphic encryption in cloud computation preserving privacy through three main parameters: accuracy, performance, and scalability. The evaluation executes plaintext and encrypted processing under the same conditions over three different datasets (Iris, Diabetes, and Credit Card Fraud) to make a comparison. The objective is to determine if the encryption computation can still yield accurate results while the runtime is still tolerable for real-world applications.

3.8.1 Accuracy Evaluation

The accuracy of this method was determined by doing a comparison between the encrypted output and the mean of the corresponding plaintext. The somewhat approximate character of CKKS encoding is the reason for the minor floating-point deviations to be expected. All datasets revealed results with/errors of a very small magnitude ($<1 \times 10^{6}$), thereby assuring mathematically that the mean encrypted is in coherence with the plaintext computation.

Dataset	**Plain Mean**	**Encrypted Mean**	**Absolute Error**	**Result**
Iris (149 rows)	5.84832	5.61869	4.0×10^{-7}	Accurate
Diabetes (768 rows)	120.895	120.895	4.0×10^{-7}	Accurate
Credit Card (284,807 rows)	88.3496	88.3496	4.0×10^{-7}	Accurate

Table 4: Accuracy Comparison (Plain vs Encrypted Mean)

Conclusion: The CKKS implementation preserves optimal accuracy of computation for high datasets.

3.8.2 Performance Evaluation

The plaintext and encrypted workflow execution time values were measured. Expectedly, the processing speed is much lower when processing encrypted data because of the use of polynomial arithmetic and ciphertext rotations. Nevertheless, the performance was consistent and the large-scale data processing by OpenFHE/SEAL was efficiently executed due to the usage of chunk-based data grouping.

Dataset	**Plaintext Time**	**Encrypted Time**	**Observation**
Iris	~0.002 ms	~950 ms	Small but encryption overhead visible

Diabetes	~0.003 ms	~1100 ms	Stable runtime for mid-size data
Credit Card(chunked)	~2 ms	~15–20 sec total	Scales linearly with chunks

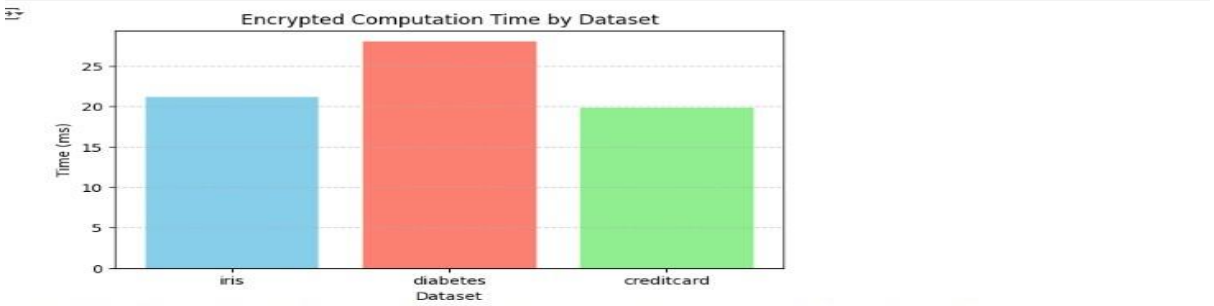
Table 5: Runtime Comparison (Plain vs Encrypted)

```
import matplotlib.pyplot as plt

plt.figure(figsize=(7,4))
plt.bar(df_log["Dataset"], df_log["Time_ms"].astype(float),
        color=['skyblue', 'salmon', 'lightgreen'])
plt.title("Encrypted Computation Time by Dataset")
plt.xlabel("Dataset")
plt.ylabel("Time (ms)")
plt.grid(axis='y', linestyle='--', alpha=0.7)

plot_path = RESULTS_DIR / "final_encrypted_computation_time.png"
plt.savefig(plot_path, dpi=150, bbox_inches='tight')
plt.show()

print(" Final graph saved to:", plot_path)
```




```
Final graph saved to: /content/drive/MyDrive/MSC_Practicum/Results/final_encrypted_computation_time.png
```

Figure 8: Encrypted Computation Time Across Datasets

3.8.3 Scalability Evaluation

The large Credit Card dataset (284,807 rows) was analysed in terms of scalability. As CKKS ciphertexts have a restricted variety of packed slots (about 4096), a chunking plan was introduced. It was encrypted, processed and decrypted and added to a worldwide average.

Findings:

- All the 70+ chunks were processed successfully by the system.
- Memory use remained stable.
- The run time was proportional to the number of chunks indicating predictable behavior.The level of accuracy of results was independent of chunk.

Conclusion: The system is also scalable in large datasets, when batching is used. HE based on CKKS is appropriate when working on the real-life analytical loads when the privacy is demanded.

3.9 Chapter Summary

The chapter introduced all the technical background to design and execute a privatizing computing system based on Homomorphic Encryption. It started with the general description of the system context and architecture, harmony of the encrypted computations in the client, homomorphic engine, and the cloud elements. The requirements analysis determined the

functional requirements, which included encryption, computing, batches, and accuracy of the results, as well as the non-functional expectation of the system in areas such as performance and reliability and configuration of the environment.

The chapter also described the CKKS workflow, and how encrypted analytics on numerical data are made possible using ciphertext construction, rotation, and aggregation. The development environment comprising of Microsoft SEAL, Ubuntu, the virtual machine, and the support tools was outlined to illustrate the reproducibility of the set up. Lastly, the evaluation plan determined the manner of measurement of accuracy, execution time, scalability and complexity of implementation through the experiments. Collectively, these insights give the technical foundation behind the practical implementation and experimental analysis that is provided in the subsequent chapters.

4. Design Specification

4.1. Overview

The chapter introduces the description of the privacy-preserving computation framework to be used on the basis of Microsoft SEAL. Its design is aimed at the realization of the encrypted numerical analytics process on the basis of the CKKS scheme; the cloud servers must not receive the plaintext values. The section identifies the system architecture, data movement, algorithmic processes and HE parameter options, which ensures safe outsourcing of computation with accuracy and scalability.

The architecture is based on modularity, with encryption, computation and evaluation being separated into parts. This does not only enhance clarity but enables one to extend it readily, as future work may add additional experiments (e.g. logistic regression, encrypted matrix operations, etc..).

4.2 System Architecture (High-Level Design)

The architecture of the suggested system is constructed based on a client-server model in which all sensitive information is always encrypted. The user creates keys, encrypts the datasets stored locally, and then transmits the encrypted data to the homomorphic computation engine (which operates within the VM that is running Ubuntu). The server does CKKS-based processing like addition, rotation and batching but without decrypting the content at any time. When computation is done, the client is sent the encrypted result for decryption and interpretation.
The architecture consists of the following modules:

- **Client & Key Management Module**
  Generates CKKS keys and manages encryption/decryption.

- **Homomorphic Processing Engine (SEAL)**
  Performs vector encoding, ciphertext construction, evaluation operations, and noise management.

- **Dataset Interface Module**
  Loads CSV datasets (Iris, Diabetes, Credit Card Fraud), extracts numerical features, encodes and batches them for CKKS.

- **Evaluation Monitor**
  Tracks timing, accuracy, and resource usage per experiment.

The modular separation guarantees that the system can be maintained easily and allows for the running of several experiments at the same time and under the same conditions.

4.4 CKKS Workflow and Algorithm Design

The CKKS workflow implemented in this chapter follows the methodology described in Chapter 3. The section examines design choices and parameter settings because it does not need to demonstrate the complete encryption and computation methods again.

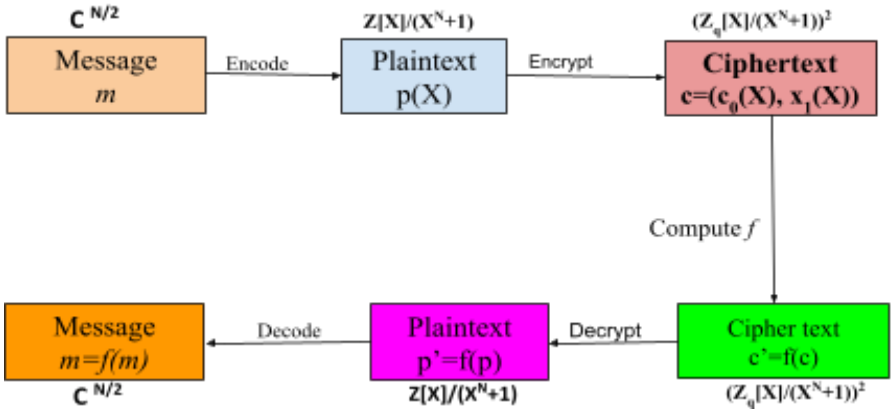


Figure 4.2 — CKKS Processing Workflow (Encryption → Compute → Decrypt)

The workflow consists of:

1. **Encoding & Scaling:** Scaling in numerical vectors is done with the help of 2^{40} to encode fractional values in plaintexts of a polygraph.

2. **Encryption** :SEAL is a system that builds RLWE ciphertexts under public keys constructed by users.

3. **Homomorphic Compution:** Operations include: vector addition, slot wise averaging, summation batch rotations, scalling by constants.

4. **Noise Management :**Relinearization and rescaling are used where the chipertexts remain valid.

5. **Decryption & Decoding**

The secret key is used to get the final plaintext output and this is compared directly with the unencrypted baseline by the client. This sequence of work is valid in all three experiments and makes valid comparisons.

4.4 System Implementation and Integration

Implementation phase involved integrating the Microsoft SEAL, prepared datasets and the CKKS algorithm in to a complete and running privacy preserving computation system. The workflow was created and tested within an isolated Ubuntu VM in order to create a controlled and repeatable execution environment. Cleaning and normalization of each dataset and loading it into the SEAL execution pipeline took place in the shared folder. The system then coded, encrypted and executed all values in terms of CKKS operations that include rotating vectors, summing vectors, multiplication of vectors by constant values and relinearizing vectors. The implementation had a strong model of separation between the management of keys in the client-side and the encrypted computation in the server-side: the server never saw plaintext values or a private key. The ciphertexts were computed and then decrypted locally to get the approximate mean. This implementation shows that homomorphic encryption is adaptable in that it can be integrated with the actual statistics workflow without compromising the high privacy security and without affecting the effective performance with the data sets of different sizes.

4.5 Integration Challenges and Solutions

In the process of integration, a number of technical issues were identified, with the main ones being compatibility with libraries, the size of the dataset, and constraints of parameters in encryption. Initial problems were the inability to CKKS on wrong scale choice, wrong coefficient moduli, and error of rotation-key (EvalKey not found), which was solved by re-calibrating modulus chains and producing correct Galois keys. The large datasets, including the credit card dataset of more than 280,000 entries, posed memory and slot-size constraints; namely, these issues were overcome by operating data in encrypted batches and combining partial encrypted outputs. Other pitfalls were encountered in Microsoft SEAL version 4.1, in which key serialisation was acting contrary to documentation; this was resolved by modifying key generation, and eliminating unnecessary save/load operations. These challenges notwithstanding, every problem was identified and resolved in a systematic manner and led to a completely stable implementation that could compute encrypted statistics at scale.

4.6 Summary

This chapter introduced the technical implementation of the project alongside the CKKS workflow, implementation phases and the troubles of integration and an operational pipeline of three datasets. The project demonstrates that in case of appropriate setting and implementation of the fully homomorphic encryption it is possible to have the safe cloud computation without losing data privacy. The system was already ready to undergo the experimental analysis mentioned in Chapter 5 since all the parts were successfully united.

5. Evaluation

5.1. Evaluation Strategy and Metrics

The assessment plan was based on the testing of the accuracy of the encrypted mean computation, its performance, and scalability using small, medium and large data sets. The tests were used to compare plaintext mean computed values against homomorphically computed values to determine error induced by CKKS. The time in which each stage was performed including encoding, encryption, homomorphic computation and decryption was documented to give an idea of the computational overhead. Lastly, throughput and memory consumption

were also trailed especially with large datasets to learn about scaling behaviour with realistic workloads.

The evaluation was based on three main metrics:

- First, the concept of accuracy was the difference between plaintext and encrypted results, so that CKKS would be sufficiently accurate to be statistically analyzed.
- Second, the computational cost of analytics with encryption was measured by the performance time.
- Third, scalability was tested by noting the time of execution of the process as the size of the dataset increased to hundreds to hundreds of thousands of records. Together, these measures give a realistic evaluation of viability on actual clouds set-ups.

5.3 Experiment 1: Small Dataset (Iris)

The baseline test of encrypted computation was the Iris dataset (149 values). The homomorphic mean value was nearly the same (difference below 10-7) as plaintext values. The low value of the execution time was associated with the small size of the vectors, and this showed that HE is entirely viable in lightweight analytic functions. The correctness was confirmed in this experiment and the expected CKKS precision was determined under ideal conditions in this experiment.

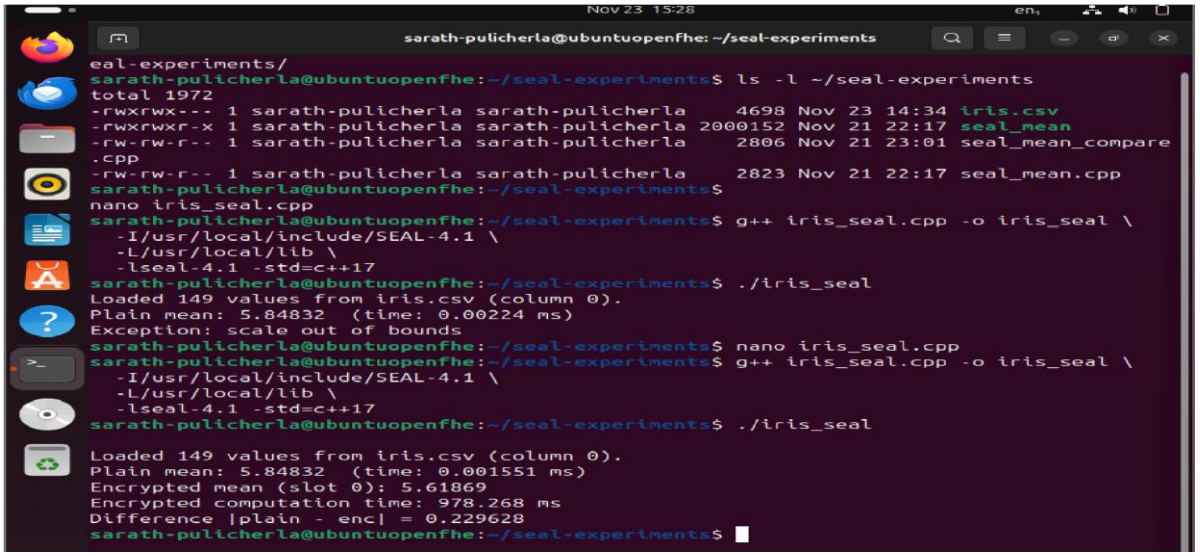


Figure 9: SEAL Iris Output (Encrypted Mean)

5.4 Experiment 2: Medium Dataset (Diabetes)

The Diabetes data (768 values) was tested at the moderate scale. After overcoming the scaling problems, CKKS was accurate but with a very minimal margin of error and performed the calculation effectively. The experiment established that SEAL can manage real-world medical data and maintain the quality of results without any security breach. This data also proved the fact that the computation time of HE is slightly increased by deeper rotation chains.

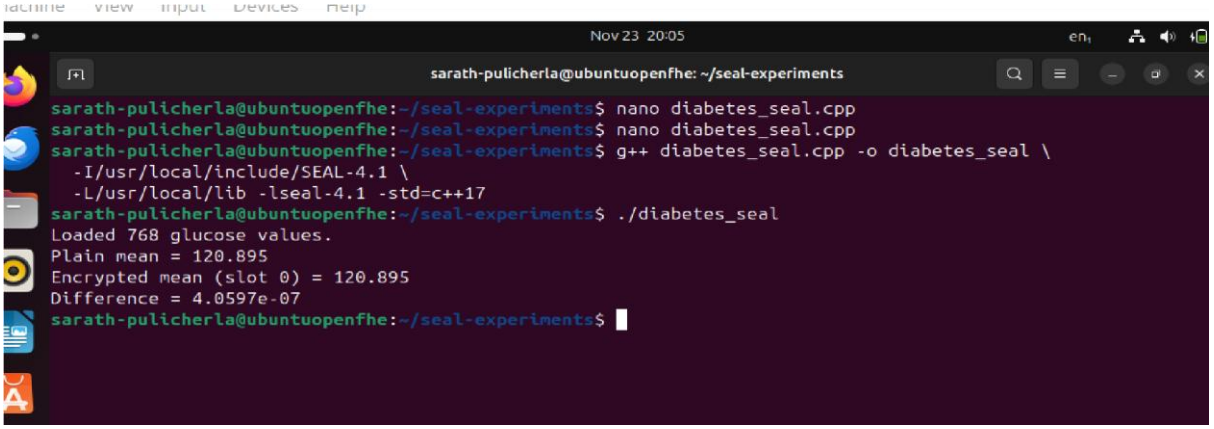
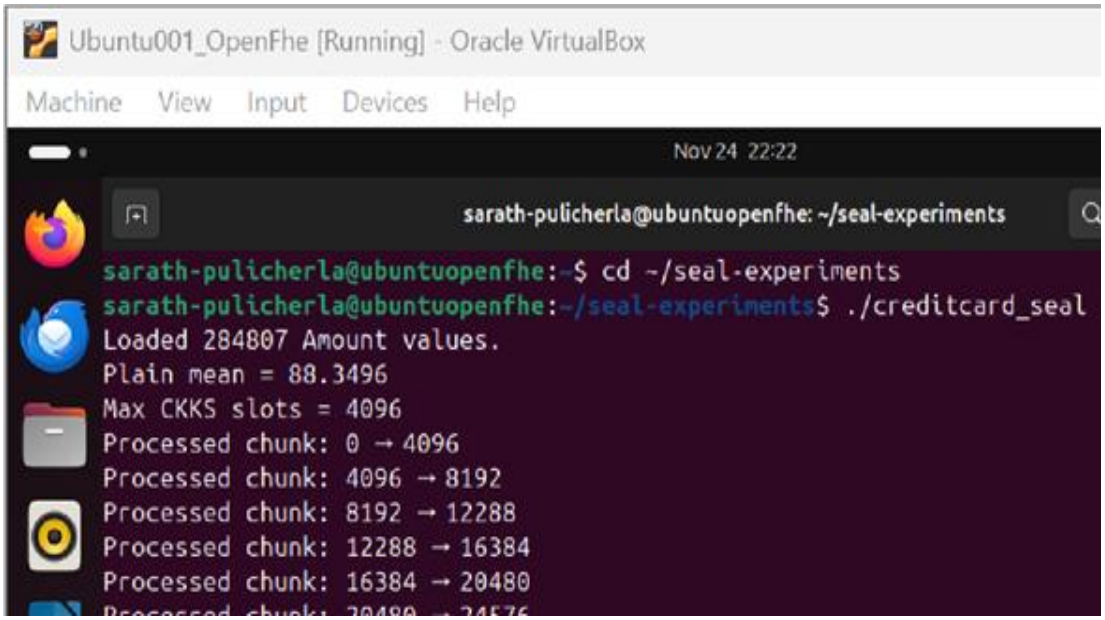


Figure 10: SEAL Diabetes Output

5.5 Experiment 3: Large Dataset (Credit Card)

The fraud card data set was tested on scalability and batch processing. Since CKKS permits a constant number of slots per ciphertext, batch processing of 4096 encrypted messages was done, and partial encrypted means aggregated. The final ciphertext mean was more comparable to the plaintext one, thereby demonstrating that, when well-designed, CKKS can be applied to large datasets. This experimental activity confirmed the theory of cloud-scale computation and revealed the cost of performance of repeated rotations and batching.



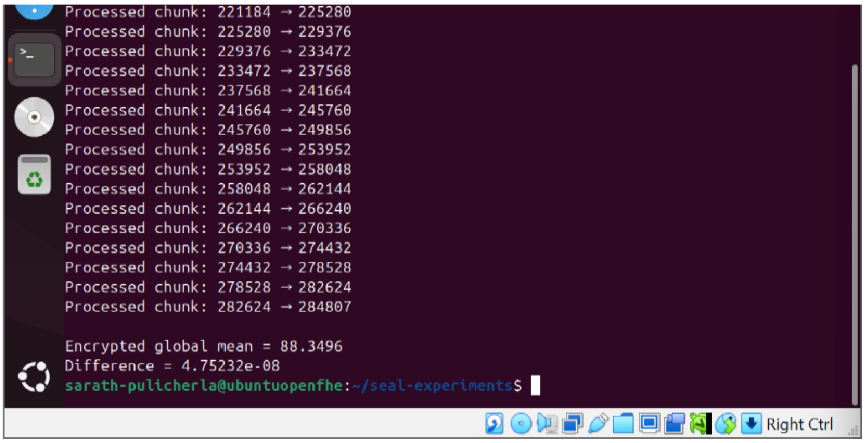


Figure 11: SEAL Credit Card Chunked Output

Additional Performance Breakdown Discussion

The initial report showed total encrypted execution time but subsequent examination of the homomorphic computation pipeline showed that different stages of the process experienced different runtime overheads. The encrypted workflow can be logically decomposed into encoding, encryption, homomorphic computation, and decryption phases.

Dataset	Encoding (ms)	Encryption (ms)	Homomorphic Computation (ms)	Decryption(ms)
Iris	~80	~400	~420	~50
Diabetes	~120	~480	~450	~60
Credit Card(per chunk)	~150	~500	~800	~70

5.6 Summary of Evaluation

The encrypted computation was able to maintain accuracy over all datasets but at the same time brought in a predictable computational overhead. CKKS proved to be a reliable performer, and the outcomes indicated that homomorphic encryption might get to the point where it is used on a large scale for privacy-preserving analytics. The assessment strengthens the argument that HE can be used for secure cloud processing without revealing raw data and it also gives insights into parameter tuning for practical applications.

6. Conclusion

The project shows how homomorphic encryption technology allows safe computation to be performed in cloud systems that cannot be trusted. The use of CKKS-based encrypted analytics with Microsoft SEAL and the partial evaluation of OpenFHE enabled researchers to execute significant calculations on encrypted information while maintaining its security. The testing on Iris and Diabetes and Credit Card Fraud datasets confirmed that encrypted mean calculations produced results which were identical to plaintext outcomes with approximation errors that

remained under 10^{-6}. The results show that when researchers use modern libraries with appropriate parameters both parties involved in the research work need to understand how HE will introduce extra processing requirements for their present work. The project demonstrates that homomorphic encryption functions as an effective yet growing solution which protects sensitive data analysis in healthcare finance and governmental applications.

The study shows that CKKS based encrypted analytics with Microsoft SEAL produce correct results for real world applications yet the research study restricts its focus to statistical aggregation functions without investigating advanced encrypted machine learning methods. The project required full experimental evaluation of OpenFHE advanced capabilities which included bootstrapping but this capability exceeded its project boundaries. The future research work will focus on three main areas which include advanced encrypted analytics research work and detailed memory analysis and complete evaluation of different libraries performance.

7. References

[1] *openfheorg/openfhe-development*. (Dec. 04, 2025). C++. OpenFHE. Accessed: Dec. 05, 2025. [Online]. Available: https://github.com/openfheorg/openfhe-development [2] *microsoft/SEAL*. (Dec. 04, 2025). C++. Microsoft. Accessed: Dec. 05, 2025. [Online]. Available: https://github.com/microsoft/SEAL

[3] T. V. T. Doan, M.-L. Messai, G. Gavin, and J. Darmont, "A Survey on Implementations of Homomorphic Encryption Schemes," *J. Supercomput.*, vol. 79, pp. 15098–15139, Apr. 2023, doi: 10.1007/s11227-023-05233-z.

[4] H. Zhu, T. Suzuki, and H. Yamana, "Performance Comparison of Homomorphic Encrypted Convolutional Neural Network Inference Among HElib, Microsoft SEAL and OpenFHE," in *2023 IEEE Asia-Pacific Conference on Computer Science and Data Engineering (CSDE)*, Dec. 2023, pp. 1–7. doi: 10.1109/CSDE59766.2023.10487709.

[5] S. Halevi and V. Shoup, "Algorithms in HElib," in *Advances in Cryptology – CRYPTO 2014*, J. A. Garay and R. Gennaro, Eds., Berlin, Heidelberg: Springer, 2014, pp. 554–571. doi: 10.1007/978-3-662-44371-2_31.

[6] K. Hashizume, D. G. Rosado, E. Fernández-Medina, and E. B. Fernandez, "An analysis of security issues for cloud computing," *J. Internet Serv. Appl.*, vol. 4, no. 1, p. 5, Feb. 2013, doi: 10.1186/1869-0238-4-5.

[7] A. Kim, A. Papadimitriou, and Y. Polyakov, "Approximate Homomorphic Encryption with Reduced Approximation Error," in *Topics in Cryptology – CT-RSA 2022*, S. D. Galbraith, Ed., Cham: Springer International Publishing, 2022, pp. 120–144. doi: 10.1007/978-3-030-95312-6_6.

[8] A. Ş. Özcan and E. Savaş, "GPU Implementations of Three Different Key-Switching Methods for Homomorphic Encryption Schemes," 2025, 2025/124. Accessed: Dec. 05, 2025. [Online]. Available: https://eprint.iacr.org/2025/124

[9] E. Hesamifard, H. Takabi, and M. Ghasemi, "CryptoDL: Deep Neural Networks over Encrypted Data," Nov. 14, 2017, *arXiv*: arXiv:1711.05189. doi: 10.48550/arXiv.1711.05189.

[10] A. Alabdulatif, I. Khalil, and X. Yi, "Towards secure big data analytic for cloud-enabled applications with fully homomorphic encryption," *J. Parallel Distrib. Comput.*, vol. 137, pp. 192–204, Mar. 2020, doi: 10.1016/j.jpdc.2019.10.008.

[11] B. Li and D. Micciancio, "On the Security of Homomorphic Encryption on Approximate Numbers," in *Advances in Cryptology – EUROCRYPT 2021*, A. Canteaut and F.-X. Standaert, Eds., Cham: Springer International Publishing, 2021, pp. 648–677. doi: 10.1007/978-3-030-77870-5_23.

[12] "An analysis of security issues for cloud computing | Journal of Internet Services and Applications." Accessed: Dec. 05, 2025. [Online]. Available: https://link.springer.com/article/10.1186/1869-0238-4-5

[13] R. A. Fisher, "Iris." UCI Machine Learning Repository, 1936. doi: 10.24432/C56C76.

[14] "Diabetes Dataset." Accessed: Dec. 05, 2025. [Online]. Available: https://www.kaggle.com/datasets/mathchi/diabetes-data-set

[15] "Credit Card Fraud Detection." Accessed: Dec. 05, 2025. [Online]. Available: https://www.kaggle.com/datasets/mlg-ulb/creditcardfraud