National College of Ireland

Configuration Manual

MSc Research Project

MSc Cloud Computing

Harinarayanan Suresh

Student ID: x22140905

School of Computing

National College of Ireland

Supervisor: Shaguna Gupta

National College of Ireland

MSc Project Submission Sheet

School of Computing

Student Name:	Harinarayanan Suresh
Student ID:	x22140905
Programme:	MSc Cloud Computing **Year:** 2023
Module:	MSc Research Project
Lecturer:	Shaguna Gupta
Submission Due Date:	31/01/2024
Project Title:	Expanding the Comparative Analysis of Privacy-Preserving Homomorphic Encryption Techniques in Cloud Computing
Word Count:	3543..... **Page Count:11**.....

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: Harinarayanan Suresh

Date: 29/01/2024

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST

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.

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).	☑

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

Configuration Manual

Harinarayanan Suresh
Student ID: x22140905

1 Introduction

This notebook, titled "Evaluating Homomorphic Encryption on Real World Datasets," is designed to provide a practical and comprehensive overview of applying homomorphic encryption techniques to various datasets. Aimed at students, academics, and professionals interested in data security and cryptography, this guide offers a hands-on approach to understanding and implementing homomorphic encryption.
Homomorphic encryption is a crucial cryptographic technique that allows for computations on encrypted data without the need to decrypt it. This capability is increasingly important in fields where data privacy is paramount, such as healthcare, finance, and personal data protection.

In this notebook, we explore the application of three homomorphic encryption techniques—CKKS, RIPPLE, and Lattice-Based—across five different datasets: MNIST, IRIS, Adult Income, Heart Disease, and Cancer Wisconsin. Each dataset presents a unique set of characteristics, making them ideal for demonstrating the versatility and effectiveness of these encryption methods.

The structure of the notebook is as follows:

1. **Data Preparation:** We begin by introducing each dataset, followed by necessary preprocessing steps to make the data suitable for encryption.
2. **Encryption Implementation:** The notebook then delves into the implementation of the CKKS, RIPPLE, and Lattice-Based encryption techniques. Each section is detailed and structured to enhance understanding and facilitate practical application.
3. **Results Analysis:** Lastly, we assess and compare the performance of each encryption technique across the datasets. This analysis aims to highlight the practical implications and efficiency of homomorphic encryption in real-world data scenarios.

This notebook is intended to be both informative and engaging, encouraging readers to interact with the content and apply the concepts learned. Our goal is to provide a clear, academically sound, and professional exploration of homomorphic encryption, making it accessible and relevant to a wide audience.

2 Required Specification

In order to effectively run and interact with the "Evaluating Homomorphic Encryption on Real World Datasets" notebook, it is important to have the appropriate hardware and software setup. This section provides a comprehensive guide on the recommended hardware specifications, necessary software, and library dependencies required to ensure an optimal experience with the notebook. These recommendations are designed to facilitate smooth execution of the encryption techniques and efficient handling of the datasets involved.

2.1 Hardware Specifications

Actual Specification for Execution Environment

The notebook "Evaluating Homomorphic Encryption on Real World Datasets" has been executed and tested on a robust and high-performance environment provided by Google Colab Pro. This setup ensures efficient handling and processing of data, especially given the computationally intensive nature of homomorphic encryption algorithms. The actual hardware specifications of the environment are as follows:

- **Backend:** Python 3 Google Compute Engine backend, augmented with TPU (Tensor Processing Unit) capabilities for enhanced computational power.
- **System RAM:** A substantial 35.2 GB of RAM, facilitating smooth handling of large datasets and complex computations.
- **Disk Space:** A generous 225.8 GB of disk space, more than sufficient for storing extensive datasets and any additional files generated during the notebook's execution.

Recommended Specifications for Users

While the notebook has been executed on a high-end platform, it is designed to be accessible and runnable on a wide range of hardware setups. For users looking to run this notebook in their environments, the following specifications are recommended:

- **Processor:** A modern multi-core processor (e.g., Intel Core i5/i7/i9 or AMD Ryzen series) to handle the computational demands of encryption algorithms efficiently.
- **Memory:** A minimum of 8 GB of RAM is recommended. However, for handling larger datasets or for more intensive data processing tasks, 16 GB or more is preferable.
- **Storage:** Adequate disk space for the datasets and any additional output files. A minimum of 50 GB of free space is recommended to ensure smooth operation. An SSD (Solid State Drive) is preferred for faster data read/write speeds.
- **Graphics Card (Optional):** While not a necessity for most encryption-related tasks, a dedicated GPU can be beneficial for specific data processing or machine learning tasks that may be part of the notebook.

These recommended specifications aim to strike a balance between accessibility and performance, ensuring that a wide range of users can effectively work with the notebook without necessitating a high-end computing environment.

2.2 Software Specifications

To ensure the successful execution of the "Evaluating Homomorphic Encryption on Real World Datasets" notebook on a local machine, certain software requirements must be met. These specifications are essential for creating an environment that can effectively handle the notebook's operations, including data processing and homomorphic encryption techniques.

- **Operating System:** The notebook is compatible with various operating systems including Windows, macOS, and Linux. Ensure that your operating system is up to date for optimal performance and compatibility.
- **Python Installation:** Python is the foundational programming language for this notebook. A Python 3.x version is required, with Python 3.6 or later recommended for better compatibility with the latest libraries and tools.

- **Jupyter Notebook:** This interactive computing environment is where the notebook will be accessed and run. It allows for a seamless integration of code execution, rich text, visualisations, and other media. Jupyter Notebook can be installed in the following ways:

  Standalone Installation: Using Python's package manager pip: `pip install notebook`

  - **Anaconda Distribution:** Installing Anaconda, a Python distribution which includes Jupyter Notebook, is an easy way to get everything set up, especially for those who are new to Python or data science. Download and install Anaconda from https://www.anaconda.com/products/individual.
- **Python Virtual Environment (Optional but Recommended):** Setting up a virtual environment is a good practice to manage dependencies and isolate the project. This can be done using `venv` or `conda` environments. A virtual environment ensures that the installation of the required libraries for this notebook does not affect other Python projects or system-wide settings.
- **Internet Connection:** A stable internet connection is necessary, especially if you're installing Python, Jupyter Notebook, or other dependencies from the internet, or if your notebook fetches data from online sources.

By adhering to these software specifications, users can create a local environment capable of running the notebook efficiently, thereby allowing for a productive and educational experience in exploring homomorphic encryption techniques on real-world datasets.

2.3 Library Dependencies

For the successful execution and exploration of the "Evaluating Homomorphic Encryption on Real World Datasets" notebook, it is essential to have specific Python libraries installed. These libraries provide the necessary tools for data manipulation, visualisation, machine learning, and homomorphic encryption techniques. Below is a detailed list of these dependencies and instructions on how to install them.

Data Handling and Visualization Libraries:

- **NumPy:** Used for numerical computations and array manipulations.
- **Pandas:** Essential for data manipulation and analysis.
- **Matplotlib:** A plotting library for creating static, interactive, and animated visualisations in Python.

  Installation Command:

```
pip install numpy pandas matplotlib
```

Machine Learning Libraries:

- **Scikit-Learn:** Provides simple and efficient tools for data mining and data analysis.
- **TensorFlow Keras:** Used for loading the MNIST dataset.

  Installation Command: `pip install scikit-learn tensorflow`

Specific Datasets:

- The notebook uses functions like load_iris and load_breast_cancer from Scikit-Learn to load the IRIS and Breast Cancer Wisconsin datasets directly.
- The MNIST dataset is loaded via TensorFlow Keras.

Note: These datasets do not require separate installation as they are part of the Scikit-Learn and TensorFlow Keras libraries.

Homomorphic Encryption Libraries:

- **Piheaan:** A library for homomorphic encryption.
- **Pyfhel:** Another Python library for homomorphic encryption, providing Python bindings for the Microsoft SEAL library. Installation Command:

```
pip install piheaan Pyfhel
```

System Utilities:

- **Sys:** A module that provides access to some variables used or maintained by the Python interpreter and to functions that interact strongly with the interpreter. It is part of the standard Python library.
- **Time:** A module for time-related tasks. Also included in the standard Python library. Note: Both `sys` and `time` are part of the standard Python library and do not require separate installation.

Ensure that these libraries are installed in your Python environment before running the notebook. This will allow you to fully engage with all aspects of the notebook, from basic data handling to implementing and understanding homomorphic encryption techniques.

3 Data and its Encryption Decryption

This section of the manual delves into the core aspect of the notebook - the practical application of encryption and decryption techniques on real-world datasets. It provides a detailed overview of the datasets used, along with step-by-step instructions on how they are encrypted and decrypted using advanced homomorphic encryption methods. This guidance is crucial for understanding the implementation nuances and effectiveness of these cryptographic techniques in data analysis.

3.1 Data Loading

In the notebook "Evaluating Homomorphic Encryption on Real World Datasets," we load and prepare five key datasets, each serving a specific purpose in demonstrating homomorphic encryption techniques.

- **MNIST Dataset**
  - Loaded using TensorFlow Keras (`mnist.load_data()`), this dataset of handwritten digits is concatenated into a single data array for analysis. The notebook includes a quick verification and visualisation of the dataset.
- **IRIS Dataset**

- The IRIS dataset, useful for pattern recognition tasks, is loaded via Scikit-Learn's `load_iris()` function. It is then converted into a DataFrame, with its structure displayed for verification.

- **Adult Income Dataset**
  - Sourced from an online CSV file, this dataset is loaded using Pandas' `read_csv()` with predefined column names. It's used for socio-economic analysis, specifically income level prediction.
- **Heart Disease Dataset**
  - Also loaded from an online source with Pandas, this dataset helps in predicting heart disease presence in patients. Special handling is done for missing values, with the dataset's structure displayed post-loading. ● **Breast Cancer Wisconsin Dataset**

    Loaded using Scikit-Learn's `load_breast_cancer()`, this dataset is vital for medical diagnostics, particularly breast cancer. It's organised into a DataFrame and displayed for a comprehensive overview.

These datasets are essential for the subsequent encryption and decryption analysis, each providing a unique context for evaluating the effectiveness of homomorphic encryption techniques.

3.2 CKKS Method

The CKKS (Cheon-Kim-Kim-Song) homomorphic encryption scheme, known for its efficiency in handling real numbers, is utilised in the notebook for both encrypting and decrypting various datasets. This sophisticated approach is critical for evaluating the practicality of applying homomorphic encryption in real-world scenarios.

CKKS Implementation:

- **Data Processing (`process_data`):** This function normalises and flattens the datasets, preparing them for the encryption process. It's a crucial step to ensure data consistency and compatibility with the CKKS scheme.
- **Encryption (`encrypt_data`):** This function takes the processed data and encrypts it using CKKS. It measures and reports key metrics like encryption time, space complexity, latency, and throughput, providing valuable insights into the encryption process's efficiency.
- **Decryption (`decrypt_data`):** Following encryption, the data is decrypted using the CKKS scheme. The decryption process is closely monitored to measure the time taken, space complexity, latency, and throughput. These metrics are essential for evaluating the performance and practicality of the CKKS method in real-world applications.
- **Batch Processing (`batch_encrypt_decrypt`):** Given the large size of the datasets, this function facilitates the processing of data in batches, allowing for more efficient encryption and decryption. It also aggregates performance metrics over multiple batches, providing a comprehensive view of the overall efficiency.

Application on Datasets

1. **MNIST Dataset:**
    - The dataset is processed, encrypted, and then decrypted in batches. Each step is evaluated for performance metrics to understand the CKKS method's efficiency on image data.
2. **IRIS Dataset:**
    - Similar to MNIST, the IRIS dataset undergoes processing, encryption, and decryption, with an emphasis on evaluating the performance of CKKS on a smaller, feature-rich dataset.
3. **Adult Income Dataset:**
    - This dataset includes categorical data, which is first encoded. The encrypted data is then decrypted, and performance metrics are analysed to assess CKKS's applicability on socio-economic data.
4. **Heart Disease Dataset:**
    - After preprocessing and encoding, the dataset is encrypted and decrypted using CKKS. Performance metrics provide insights into the method's effectiveness for healthcare data.
5. **Cancer Wisconsin Dataset:**

- The focus here is on processing medical diagnostic data for encryption and decryption, with a thorough evaluation of performance metrics to determine the CKKS method's suitability for sensitive health data.

In each case, the CKKS method demonstrates its capability to handle various data types, from images to numerical and categorical data, showcasing its potential for broad applications in data privacy and security. The notebook not only implements these techniques but also provides an in-depth analysis of their performance, offering valuable insights for researchers and practitioners in the field of data security.

3.3 RIPPLE Method

The notebook implements the RIPPLE homomorphic encryption scheme, adept at handling both real and integer numbers. This method is pivotal for examining the feasibility of homomorphic encryption across various data types and scenarios.

RIPPLE Implementation

- **Data Processing (`process_data`):** This function is crucial for normalizing and formatting the datasets, making them suitable for the RIPPLE encryption process.
- **Utility Functions:** Includes functions like `crange`, `discrete_gaussian`, `discrete_uniform`, which are essential for polynomial operations within the RIPPLE scheme.
- **RIPPLE Class:** Encapsulates the core functionalities of the RIPPLE encryption scheme, including key generation (`generate_keys`), encryption (`encrypt`), and decryption (`decrypt`). It also includes methods for addition and multiplication of encrypted data, demonstrating the homomorphic properties.
- **Batch Processing (`batch_encrypt_decrypt`):** Handles encryption and decryption in batches, which is particularly beneficial for processing large datasets efficiently. This function also aggregates performance metrics, offering a comprehensive view of the method's efficiency.

Application on Datasets

1. **MNIST Dataset:**
   - Preprocessed and encrypted using RIPPLE, with performance metrics for encryption recorded.
   - The encrypted data is then decrypted, and decryption metrics are analysed.
2. **IRIS Dataset:**
   - Undergoes a similar process of RIPPLE encryption and decryption.
   - Performance metrics are evaluated to understand the method's efficiency on a smaller dataset.
3. **Adult Income Dataset:**
   - Includes a preprocessing step for encoding categorical data.
   - Post-encryption and decryption, performance metrics are assessed to gauge the method's applicability to socio-economic data.
4. **Heart Disease Dataset:**

- After preprocessing and encoding, the dataset is encrypted and decrypted using RIPPLE.
- Performance metrics are recorded to evaluate the scheme's effectiveness for healthcare data.

5. **Cancer Wisconsin Dataset:**
   - Focuses on encrypting and decrypting medical diagnostic data using RIPPLE. Performance metrics are thoroughly evaluated to ascertain the method's suitability for sensitive health data.

In each scenario, the RIPPLE method adeptly handles different data types, showcasing its versatility and potential in data privacy and security applications. The notebook effectively implements these techniques and provides an in-depth analysis of their performance, offering valuable insights for those in the field of data security and encryption.

3.4 Lattice-Based Homomorphic Encryption

The notebook incorporates a Lattice-Based Homomorphic Encryption approach, utilising the Pyfhel library. This method leverages lattice cryptography for secure data processing, enabling computations on encrypted data. It's instrumental in exploring the practicality of lattice-based encryption in diverse data scenarios.

Lattice-Based Homomorphic Encryption Implementation

- **Data Processing (`process_data`):** This function prepares the datasets for encryption, ensuring they are in the correct format. It flattens image data and converts dataframes into numpy arrays.
- **Encryption Function (`encrypt_data`):** Encrypts data using the lattice-based approach. It measures and reports key metrics like encryption time, space complexity, latency, and throughput.
- **Decryption Function (`decrypt_data`):** Decrypts the encrypted data and evaluates the decryption process, reporting similar metrics as encryption.
- **Batch Processing (`batch_encrypt_decrypt`):** Facilitates handling of large datasets by processing data in batches, both for encryption and decryption. This function also calculates average performance metrics across batches for a comprehensive analysis.

Application on Datasets

1. **MNIST Dataset:**
   - The dataset is processed, encrypted, and decrypted using the lattice-based method. Each step's performance is evaluated for efficiency metrics, providing insights into the method's applicability to image data.
2. **IRIS Dataset:**
   - Processes, encrypts, and decrypts the IRIS dataset, with performance metrics evaluated to understand the efficiency of lattice-based encryption on smaller, feature-rich datasets.
3. **Adult Income Dataset:**

- Includes preprocessing for encoding categorical data. The encrypted data is then decrypted, with performance metrics analysed to assess the method's suitability for socio-economic data.

4. **Heart Disease Dataset:**
   - After preprocessing and encoding, the dataset is encrypted and decrypted. Performance metrics provide insights into the method's effectiveness for healthcare data.
5. **Cancer Wisconsin Dataset:**
   - Processes medical diagnostic data for encryption and decryption, with thorough performance metrics evaluation to determine the method's suitability for sensitive health data.

In each case, the Lattice-Based Homomorphic Encryption method demonstrates its capacity to handle various data types, showcasing its potential for broad applications in data privacy and security. The notebook effectively implements these techniques and provides an in-depth analysis of their performance, offering valuable insights for researchers and practitioners in the field of data security.

3.5 Technique Comparison by Dataset

This section presents a comparative analysis of the three encryption techniques—CKKS, RIPPLE, and Lattice-Based Homomorphic Encryption—across various datasets. We evaluate and contrast their performance in terms of time complexity, space complexity, latency, and throughput, offering insights into the efficiency and suitability of each method for different types of data. This comparison is pivotal in understanding the trade-offs and advantages unique to each encryption technique when applied to datasets like MNIST, IRIS, Adult Income, Heart Disease, and Cancer Wisconsin.

The performance of CKKS, RIPPLE, and Lattice-Based encryption methods is analysed across multiple datasets—MNIST, IRIS, Adult Income, Heart Disease, and Cancer Wisconsin—using bar plots that focus on key metrics: time complexity, space complexity, latency, and throughput.

- **Time Complexity Bar Plot**
  - **Setup:** Side-by-side bars represent encryption and decryption times for each technique across all datasets.
  - **Data:** Average encryption (encryption_times) and decryption times (decryption_times) are calculated and displayed for each method and dataset.
  - **Visualisation:** Titled 'Encryption and Decryption Time Comparison', plots for each dataset use logarithmic y-axes for clear time complexity comparison.
- **Space Complexity Bar Plot**
  - **Setup:** Displays space complexity during encryption and decryption for each method across datasets.
  - **Data:** Space complexities (encryption_space and decryption_space) are calculated and plotted, highlighting memory usage.
  - **Visualisation:** Titled 'Encryption and Decryption Space Comparison', these plots visualise memory requirements for each dataset and encryption technique.
- **Latency Bar Plot**

    - **Setup:** Shows latency of each method during encryption and decryption across datasets.
    - **Data:** Latency values (encryption_latency and decryption_latency) are displayed for each dataset and technique.
    - **Visualisation:** Titled 'Encryption and Decryption Latency', these plots assess the responsiveness of each encryption method for each dataset.
- **Throughput Bar Plot**
    - **Setup:** Visualizes throughput of each encryption method across different datasets.
    - **Data:** Throughput values (encryption_throughput and decryption_throughput) are calculated and plotted, demonstrating data processing efficiency.
    - **Visualisation:** Titled 'Encryption and Decryption Throughput', these plots compare the data processing capabilities of each technique for each dataset.
- **Combined Analysis**
    - This comprehensive approach provides a holistic view of each encryption method's performance across diverse datasets.
      By comparing all datasets side-by-side, we can draw broader conclusions about each method's overall efficiency, scalability, and suitability for various types of data.
    - This analysis is crucial for understanding the strengths and limitations of each encryption method, allowing for informed decisions when selecting the most appropriate technique for specific data processing needs.

The bar plots collectively offer a nuanced understanding of how each encryption method performs under different data scenarios, guiding users in choosing the right technique for their specific use case and dataset.

References

1. *NumPy: NumPy contributors. NumPy: A fundamental package for scientific computing with Python. Available at: https://numpy.org/*

2. *Pandas: McKinney, W. Pandas: A Foundational Python Library for Data Analysis and Statistics. Available at: https://pandas.pydata.org/*

3. *Matplotlib: Hunter, J.D. Matplotlib: A 2D Graphics Environment. Computing in Science & Egineering, 2007. Available at: https://matplotlib.org/*

4. *TensorFlow Keras: Chollet, F. et al. Keras: The Python Deep Learning API. Available at: https://keras.io/*

5. *Scikit-Learn: Pedregosa, F. et al. Scikit-learn: Machine Learning in Python. Journal of Machine Learning Research, 2011. Available at: https://scikit-learn.org/stable/*

-

6. *Cheon-Kim-Kim-Song (CKKS) Homomorphic Encryption Scheme: Cheon, J.H., Kim, A., Kim, M., and Song, Y. Homomorphic Encryption for Arithmetic of Approximate Numbers. Advances in Cryptology – ASIACRYPT 2017.*