Project 3: Privacy Through Homomorphic Encryption

Privacy-Preserving Salary Audit

Mathias Welz

December 14, 2025

1 Introduction

The objective of this project is to explore the concepts of Homomorphic Encryption (HE) and apply them to a practical scenario where data privacy is paramount. By utilizing libraries such as `phe` (Partial Homomorphic Encryption) and `tenseal` (Approximate Homomorphic Encryption), I demonstrate how computations can be performed on encrypted data without ever revealing the underlying plaintext.

2 Step 1: Problem Definition and Dataset

2.1 The Scenario: Privacy-Preserving Salary Audit

The Core Problem: A corporation (Data Holder) needs to perform a statistical audit of its employee payroll to calculate the total expenditure (Sum) and the average salary (Mean). To ensure neutrality, this audit is outsourced to an external Cloud Auditor (Data Analyzer). However, the corporation cannot reveal individual employee salaries to the Auditor due to privacy risks.

The Legal Constraint: Regulations such as GDPR prohibit the sharing of unencrypted, personally identifiable financial data. The Data Holder requires a solution where the Auditor can perform mathematical operations (Addition and Division) on the data while it remains encrypted.

The Solution: I implement a Homomorphic Encryption workflow. The Data Holder encrypts the salaries; the Data Analyzer computes the sum and mean on the ciphertexts; and the Data Holder decrypts the result.

2.2 Dataset Creation

The dataset was generated synthetically to simulate a realistic payroll file. A Python script, `dataset_gen.py`, was created to generate a CSV file named `salaries.csv`.

- **Content:** Random integer values ranging from 30,000 to 120,000, representing annual salaries.
- **Structure:** The file contains two columns: `Employee_ID` and `Salary`.

Only the `Salary` column is processed. To test scalability, I generated datasets of varying sizes: **1,000, 5,000, and 13,000 records**.

3 Step 2 & 3: Applying Homomorphic Encryption

I implemented two distinct schemes to compare their efficiency and mathematical properties.

3.1 System Architecture & Demonstration Workflow

The implementation is designed to demonstrate the complete lifecycle of secure computation, divided into two roles:

1. **Data Holder (Encryption):** The Data Holder loads the `salaries.csv`, generates the public/private key pairs, and encrypts the raw data. This step converts sensitive integers into ciphertexts (Paillier) or a secure vector (CKKS).

2. **Data Analyzer (Computation):** The Analyzer receives only the encrypted files. It performs the statistical operations blind:
    - *Sum:* Adds ciphertexts together (homomorphic addition).
    - *Mean:* Multiplies the encrypted sum by the scalar $1/N$.
3. **Data Holder (Decryption):** Finally, the Holder receives the encrypted result, uses the private key to decrypt it, and interprets the final mean salary.

3.2 Scheme 1: Paillier (via `phe`)

The Paillier cryptosystem is a partial homomorphic encryption (PHE) scheme [1]. I selected it for its additive homomorphic property, allowing exact summation.

3.2.1 Mathematical Foundation

As illustrated in Figure 1, the scheme relies on the decisional composite residuosity assumption.

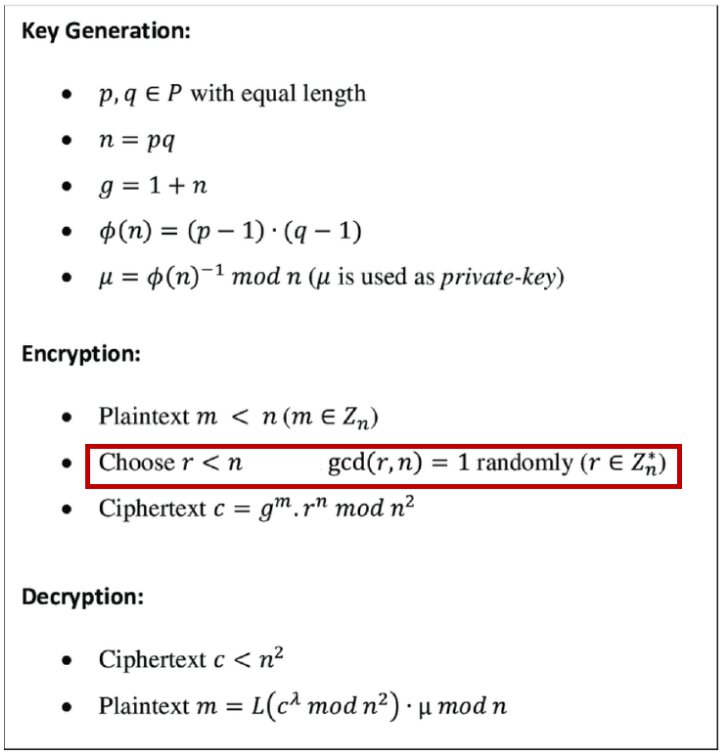

Figure 1: Key Generation, Encryption, and Decryption in Paillier

3.2.2 Homomorphic Addition Scenario

The Analyzer multiplies ciphertexts to implicitly compute the sum: $D(E(m_1) \cdot E(m_2)) = m_1 + m_2$. To compute the mean, the `phe` library encodes $1/N$ as a fixed-point integer for scalar multiplication.

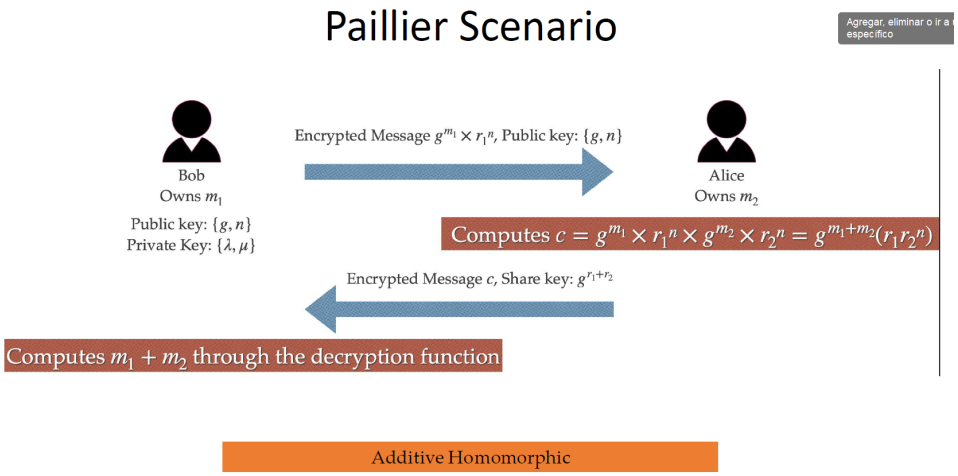

Figure 2: The Additive Homomorphic Property of Paillier

3.3 Scheme 2: CKKS (via `tenseal`)

CKKS is a leveled homomorphic encryption scheme for approximate arithmetic. I utilized the `tenseal` library to implement this using SIMD (Single Instruction, Multiple Data) operations.

3.3.1 Vectorized Operations

CKKS allows packing the entire dataset into a single ciphertext vector.

- **Efficiency:** Reduces complexity from $O(N)$ to effectively $O(1)$ vector operations.
- **Approximation:** I configured the `global_scale` to 2^{40} to ensure noise remains negligible.

4 Step 4: Results and Comparison

I executed both schemes on datasets of 1,000, 5,000, and 13,000 entries.

4.1 Execution Results

Figures 3, 4, and 5 display the terminal outputs, validating the correct decryption of the mean salary across all tests.

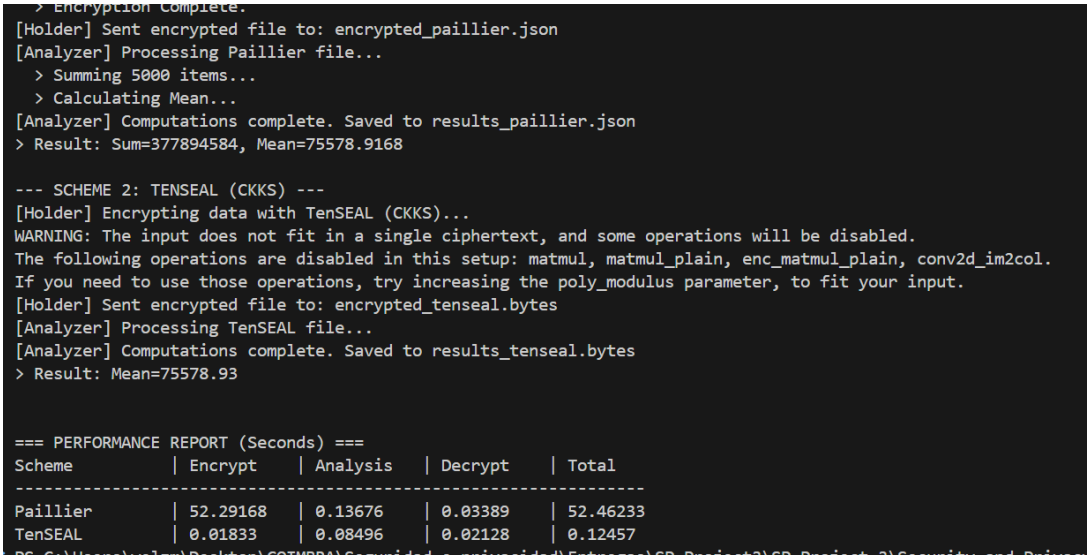

Figure 3: Execution Output for 1,000 Records

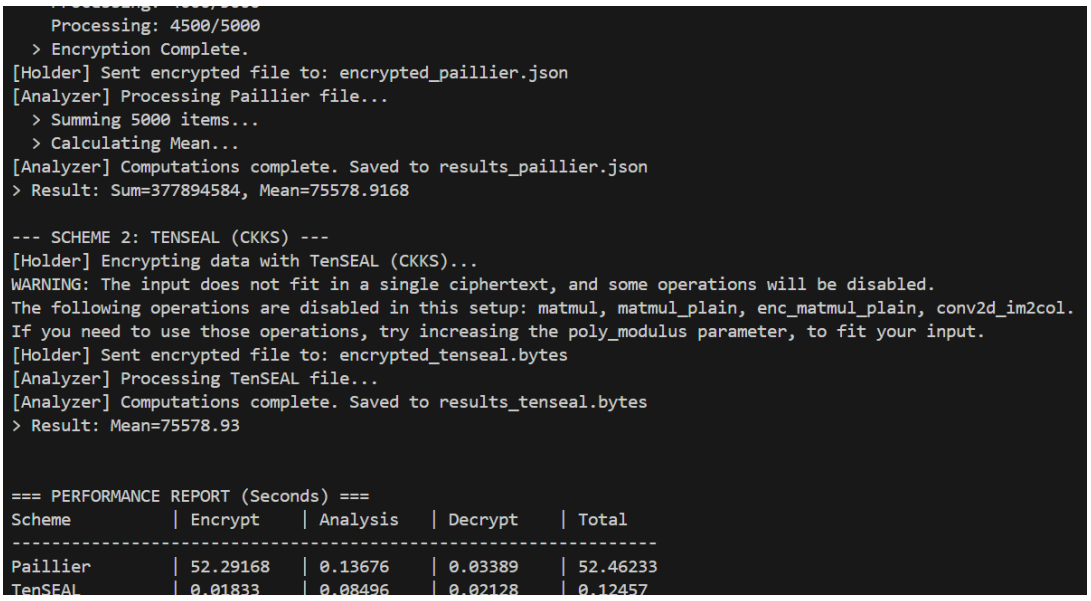

Figure 4: Execution Output for 5,000 Records

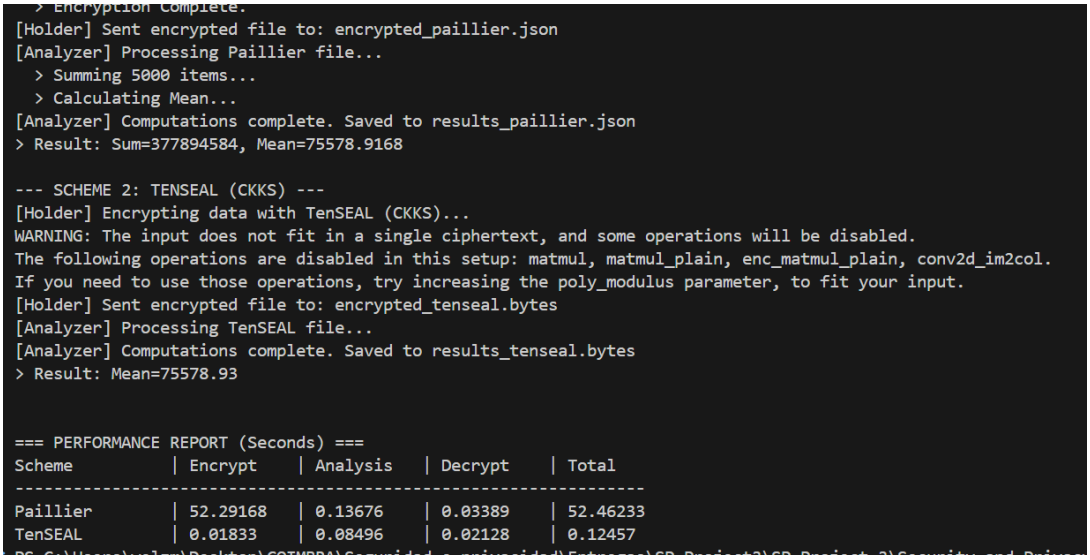

Figure 5: Execution Output for 13,000 Records

4.2 Performance Table

The following table summarizes the execution times. Note the dramatic difference as data volume increases.

Scheme	Dataset Size	Encrypt (s)	Analysis (s)	Total (s)
Paillier	1,000	10.72	0.08	10.83
TenSEAL	1,000	0.01	0.11	0.14
Paillier	5,000	52.29	0.13	52.46
TenSEAL	5,000	0.02	0.08	0.12
Paillier	13,000	132.22	0.28	132.53
TenSEAL	13,000	0.03	0.09	0.15

Table 1: Scalability Comparison (1k, 5k, 13k Records)

4.3 Discussion

The results highlight a trade-off between precision and speed:

- **Paillier Scalability (Linear):** As expected, Paillier's execution time grows linearly with the dataset. Processing 13,000 records took over 2 minutes (132s). This confirms that while Paillier is excellent for small, exact computations, it struggles with "Big Data" volumes due to its iterative nature.
- **CKKS Efficiency (Vectorized):** TenSEAL's performance remained nearly flat, increasing from 0.14s to only 0.15s despite a 13x increase in data. This proves the power of SIMD operations, making CKKS the superior choice for large-scale analytics.
- **Correctness:** In all cases, the decrypted means were mathematically consistent (e.g., Paillier: 75147.13 vs TenSEAL: 75147.14 for 13k records), verifying that both schemes correctly preserved the data utility.

5 Conclusion

This project demonstrated a fully functional privacy-preserving audit system. While Paillier provided exact integer arithmetic suitable for strict accounting, the CKKS scheme (TenSEAL) demonstrated orders-of-magnitude better performance for large datasets, making it the practical choice for cloud-based analytics.

References

[1] CSIRO (Data61), *API Documentation — python-paillier*, Available at: `https://python-paillier.readthedocs.io/en/develop/phe.html` (Accessed: December 14, 2025).