Security & Privacy

Till Müller

Student Number: 2025181168

UNIVERSIDADE D
COIMBRA

Privacy Through Homomorphic Encryption

16. November 2025

Supervisor: Prof. Dr. Naghmeh Ramezani Ivaki

1 Scenario

In this project, a fictional but realistic scenario is used to demonstrate privacy-preserving analytics with Homomorphic Encryption (HE). The setting is a futuristic crisis situation in which parts of a country are affected by radio active contamination, natural disasters, or armed conflicts. Some regions are considered unsafe due to radiation exposure or enemy activity.

To assess these dangerous areas, autonomous drones are deployed to collect critical information directly from the (Battle)-field. These drones gather important data points, such as radiation levels and indicators of hostile activity, which are necessary for evaluating risks and supporting strategic decisions. Because this information is highly sensitive, it must be protected throughout the entire processing pipeline.

The central idea of the scenario is that all collected data is protected before any external processing takes place. The drones do not send their measurements directly to the cloud computing platform. Instead, all data is first transmitted to a trusted command center. This command center can be understood as a military base or, more generally, as a strategic coordination unit in a crisis or disaster response situation.

At this command center, the incoming drone data is prepared for secure analysis. The encryption system is initialized and all sensitive measurements are encrypted inside this trusted environment. Only encrypted data is allowed to leave the command center. This ensures that no sensitive information is exposed, even if the next processing stage is not fully trusted.

The encrypted data is then sent to an external server center or cloud infrastructure. This cloud is assumed to be untrusted, but it provides strong computational power. It is able to perform complex and large-scale computations on the encrypted data, which would be difficult or too slow to execute locally. During this process, the cloud never gains access to the original values or any secret keys.

After the computation is completed, the encrypted results are returned to the command center, where they are decrypted and evaluated to obtain meaningful insights. This workflow enables fast and large-scale data analysis even in hostile or uncertain environments, while ensuring that sensitive information remains protected at every stage. The scenario highlights how Homomorphic Encryption makes it possible to rely on external computing resources without exposing critical data, and demonstrates a practical path toward secure decision support under extreme conditions.

The overall workflow is structured as follows:

- **Data Collection:** The drones collect two types of measurements: Floating-point radiation levels (for example in mSv/h) and integer values representing enemy presence in different zones.
- **Forward Base (Data Holder):** At a trusted military base, initializes the encryption system and generate all necessary keys. The measurements collected by the drones are encrypted before transmission. In addition, calculation parameters such as thresholds, weights, or constants are also encrypted, so that the analytic logic itself remains confidential.
- **Secure Cloud (Data Analyzer):** An untrusted cloud server receives only encrypted data together with public encryption parameters. The cloud performs all required computations directly on the ciphertexts using homomorphic operations. At no point the cloud has access to secret keys or plaintext values.
- **Verification and Decision:** After the computation, the encrypted results are sent back to the Data Holder (Command Center). There, the results are decrypted, verified for correctness, and interpreted to assess radiation safety, threat levels, or possible safe routes.

This setup clearly separates responsibilities. Full control over all secrets and plaintext data remains within the trusted environment, while the cloud is restricted to operating only on

encrypted information. This makes the scenario a practical and intuitive example of how Homomorphic Encryption can be applied in sensitive real-world situations.

2 Datasets

To enable reproducible experimentation, a synthetic data set was created. The data consists of 1,500 generated entries and is produced by the `scenario_data.py` script. This script allows easy adjustment of the number of data points, making it possible to test different data sizes and evaluate scalability behavior.

The generated data is automatically stored in the `data/` directory when the script is executed. Separate data sets are created for the two encryption schemes, reflecting their different data types:

- `data/raw_data_ckks.json`: Floating-point radiation measurements used for CKKS-based analytics.
- `data/raw_data_bfv.json`: Integer values that represents enemy presence counts used for BFV-based analytics.
- `data/expected_values.json`: Expected summary values, such as average radiation levels, total enemy counts, and the number of measurements, which are used later for result verification.

These data are small but sufficient to demonstrate key behaviors of HE while avoiding sensitive real-world information.

3 System Roles and Files

- **Data Holder (Notebook: `data_holder.ipynb`):** This component represents the trusted command center or military base in the scenario. It is responsible for generating all cryptographic contexts, including public and private keys for CKKS and BFV schemes. The Data Holder encrypts all collected measurements as well as calculation parameters. After the Analyzer finishes the computations, the Data Holder decrypts the results and verifies their correctness. This role ensures that all sensitive information remains fully under control.
- **Analyzer (Notebook: `data_analyzer.ipynb`):** This component simulates an untrusted cloud computing platform. It receives only the public contexts and encrypted data from the Data Holder. The Analyzer performs all necessary computations on the encrypted inputs and writes the encrypted outputs back to the Data Holder. At no point does it gain access to the plaintext or any secret keys.
- **Utilities (`utils.py`):** A small helper module for file handling. It reads and writes serialized keys, contexts, encrypted data, and results. The Data Holder uses it to save encrypted inputs and parameters, while the Data Analyzer uses it to load these inputs and save the computed outputs. This makes exchanging encrypted artifacts between the two parts of the project simple and reliable.
- **Keys Folder (`keys/`):** This folder stores the public and private keys used for encryption and decryption. The keys are generated by the Data Holder and are required to securely encrypt inputs and later decrypt the results.

4 Homomorphic Encryption Schemes

The project uses two homomorphic encryption schemes to handle different types of data securely.

- **CKKS (Approximate Arithmetic):** Works on real numbers using floating-point values. It is applied to continuous radiation measurements collected by drones. The Analyzer can compute averages, apply encrypted weights, and calculate safe-route scores. Because CKKS is approximate, decrypted results may have small numerical errors, but these remain within an acceptable range for analysis.
- **BFV (Exact Integer Arithmetic):** Works on integers and is used for counting discrete events, such as enemy presence in different zones. BFV ensures that sums and other aggregations are exact. After decryption, the results match the expected values precisely, without any rounding errors.

Using both schemes demonstrates how different types of data can be securely processed. CKKS is suitable for continuous measurements where small errors are acceptable, and BFV is used for counts that require exact results. All calculation parameters are encrypted, keeping the logic hidden from the Analyzer, and results can be verified after decryption.

5 Code Workflow and System Operation

The workflow functions as follows:

- **Key Generation and Parameter Encryption (Data Holder):** The Data Holder, representing a trusted command center, generates CKKS and BFV contexts, including public and private keys. The BFV secret key is about 3.9 MB, and the CKKS secret key is around 908 KB. Calculation parameters such as weights, thresholds, divisors, and constants are encrypted so that all analytic logic remains hidden. Data and parameters are always encrypted before being transmitted.
- **Homomorphic Computation (Analyzer):** The Analyzer receives only encrypted data and public contexts. It performs computations directly on ciphertexts. For CKKS, floating-point radiation measurements are summed, weighted, and used to compute a safe-route score. For BFV, integer enemy counts are summed exactly across all zones. The Analyzer never has access to plaintext values or secret keys.
- **Result Decryption and Verification (Data Holder):** Encrypted results are returned to the Data Holder, where they are decrypted and compared with expected values for verification. CKKS results are checked for small numerical errors within an acceptable tolerance, while BFV results are verified to match exactly.

The code itself is clearly documented, providing detailed explanations of each step, including encryption setup and computation logic. This allows the workflow to be understood and reproduced.

5.1 Outputs

All encrypted artifacts are stored in the `outputs/` folder and can be divided into inputs, parameters, and results:

- **Encrypted Inputs:**
  - `outputs/radiation_encrypted.txt`: CKKS vector of radiation measurements

  - `outputs/enemy_encrypted.txt`: BFV vector of enemy counts

- **Encrypted Parameters:**
  - `outputs/weight_radiation_encrypted.txt`: encrypted radiation weight
  - `outputs/weight_enemy_encrypted.txt`: encrypted enemy weight
  - `outputs/divisor_encrypted.txt`: encrypted divisor $1/n$ for average calculation
  - `outputs/threshold_radiation_encrypted.txt`: encrypted radiation threshold (50.0)
  - `outputs/threshold_enemies_encrypted.txt`: encrypted enemy threshold (10.0)
  - `outputs/safe_route_constant_encrypted.txt`: encrypted constant (100.0) for the safe-route score

- **Encrypted Results:**
  - `outputs/radiation_average_encrypted.txt`: CKKS radiation average
  - `outputs/radiation_score_encrypted.txt`: CKKS weighted radiation score
  - `outputs/safe_route_score_encrypted.txt`: CKKS safe-route score
  - `outputs/enemy_sum_encrypted.txt`: BFV total enemy count

- **Keys (stored in `keys/` folder):**
  - `ckks_secret.txt` and `bfv_secret.txt`: secret keys for the Data Holder (used for decryption)
  - `ckks_public.txt` and `bfv_public.txt`: public keys for the Analyzer (used to perform computations on encrypted data)

6 How to Run

1. Run `scenario-data.py`: Generate the synthetic data sets for CKKS (radiation) and BFV (enemy counts).
2. Open `data_holder.ipynb`: Generate keys, encrypt data and parameters, and save artifacts.
3. Open `data_analyzer.ipynb`: Load public contexts and encrypted inputs, run CKKS and BFV workflows, and save encrypted results.
4. Return to `data_holder.ipynb`: Load encrypted results, decrypt, and verify against `expected_values.json`.

Conclusion

This project shows how privacy-preserving analytics can be used in a hostile or disaster scenario using homomorphic encryption. Drones collect radiation levels as floating-point numbers and enemy counts as integers. All measurements are encrypted before processing so that neither the data nor the decision rules are revealed. CKKS is used for approximate computations on continuous sensor data, such as averages, weighted scores, and safe-route metrics. BFV is used for exact integer computations, such as counting enemies. The Data Holder encrypts all parameters, including weights, thresholds, divisors, and constants, to keep the rules and priorities confidential. The Analyzer works only on encrypted data and cannot see the original values. After computation, the Data Holder decrypts and verifies the results. CKKS results are checked for small numerical errors, while BFV results are verified for exact correctness. This workflow provides a secure way to support situational assessment and routing decisions while keeping all data and logic fully confidential.