Homework 8

Exercise 1: [PET] Homomorphic Encryption for ML Model Inference

A company named SecureHealth provides an API to query what they call a "Health AI" in a private way. According to their website, the company uses some complicated "novel medical AI method", and sells predictions using their model. A customer can send a feature vector representing different health-related features of a patient as input, and get back a score of how likely is some given diagnosis. Your institution has paid a lot of money to get access to an API to query this model in a privacy-preserving way. That is, without SecureHealth being able to see the feature vector that is submitted to its model, nor the computed output score.

In order to achieve that, SecureHealth relies on homomorphic encryption. Homomorphic encryption is a malleable form of encryption which allows arithmetic operations to be performed on encrypted data without having to decrypt. In particular, they use the Paillier Encryption Scheme (see lectures).

Setup

We simulate the SecureHealth service using a docker container: This container runs the SecureHealth service as a HTTP server, which you can start with:

```
docker run --rm -p 8000:8000 com402/hw8
```

Then, you can access the service by making HTTP requests to `http://localhost:8000` (there is no homepage, visiting the page with your browser will yield a 404).

ML model

The **plaintext** predictive model takes a vector of ten features (extracted from the patient's data) and outputs a score (probability of disease):

- The input $\mathbf{x}$ is a feature vector of 10 positive, normalized numbers in the range $[0, 1]$.
- The output y score is a probability, hence a positive number in the range $[0, 1]$

The scoring function is a linear combination of the input features, computed as $y = \mathbf{x} \cdot \mathbf{w} + b$, where

- $\mathbf{w}$ is the coefficient vector of positive numbers in $[0, 1]$
- b is a bias term in $[0, 1]$

In our case, the interface accepts **encrypted** inputs, computes the score using **encrypted** arithmetic and yields **encrypted** outputs.

Cryptographic parameters

The prediction service expects the following cryptographic parameters for the public key and ciphertexts.

Paillier public key:

- Modulus (N) bit-length: 2048
- Generator (g): $N + 1$

Paillier ciphertexts:

- Fixed-point encoding precision: 16 bits

HTTP interface

Here are the details of their HTTP interface.

- `POST /prediction`
  - Request body, a JSON-object with the following fields:
    - `pub_key_n`: the Paillier public key modulus N, an integer in the base-10 representation
    - `enc_feature_vector`: the encrypted feature vector, a JSON-array of integers in the base-10 representation
  - Responses:
    - 200 (json):
      - `enc_prediction`: the encrypted prediction, an integer in the base-10 representation
    - `40x` (text):
      - error message

Note: The `requests.post([URL], json=[JSON object])` function sends a post request with the appropriate heads and JSON body.

Task

Your first task is to implement a client to query the SecureHealth model API.

You may find the python-paillier library useful. It is well documented, and its *docstring* documentation (`help(paillier)`) is really useful in understanding both the implementation details and the Paillier cryptosystem usage.

[Optional] Even though the server is provided to you, understanding the homomorphic computation it performs is part of the homework. You might want to try to implement this part yourself, as it is key to understand how to work with the Paillier scheme for floating point values. The english wikipedia article is a good starting point.

Since the Paillier encryption scheme supports integer plaintext messages, floating point values need to be encoded as integers before encryption, and decoded back to floating points value after decryption. This encoding is usually done by scaling the plaintext up by a constant, and scaling it down at decoding. Indeed, the higher the constant, the higher the *precision*, but this has some other side-effect too. You will need to investigate the details of these side effect as they are key for the correctness of the output, and is part of this homework.

Note: the weight vector $\mathbf{w}$ that the server uses is also encoded in the fixed-point representation. You should keep track of how the server's computation changes the scaling constant.

Tip: You can either convert reals into their fixed-point representations manually, or use the provided `precision` and `exponent` parameters of the Paillier library.

Tip 2: The code to encrypt/query/decrypt is rather straightforward, and not really what we teach you with that exercise. But one of the main important aspect, that you need to understand, is how the scaling works, and what the parameters `precision` and `exponent` are. Get some paper, you'll need it.

You can use the following test vector to test your implementation (given that your client implements the API querying logic in `query_pred`):

```
assert 2**(-16) > abs(query_pred([0.48555949, 0.29289251, 0.63463107,
                                  0.41933057, 0.78672205, 0.58910837,
                                  0.00739207, 0.31390802, 0.37037496,
                                  0.3375726 ]) - 0.44812144746653826)
```

Exercise 2: [attack] Prediction-as-a-Service Model Stealing

You found an interview of a former SecureHealth chief data scientist. You realized that their "AI" actually uses a linear regression (hence, the linear scoring function…). Having paid a lot of money to query "an advanced AI", you feel somewhat betrayed and decide to end the subscription. But before that, you want to perform a *long term, unilateral, model sharing* (i.e., steal it).

Your task is to write a program that extracts the model parameters using the prediction API. Also think about a way of validating that the extracted parameters are correct.

Notes

As you might have already noticed, the server implements a (very basic) protection against a (very basic) attack.