Homomorphic Encryption Demonstration Page

Welcome to the Homomorphic Encryption Demonstration Page. Here, you'll get to experience Homomorphic Encryption (HE) being used to perform simple equations on encrypted data, without the need to decrypt the inputs first.

Pre-requisites

This demonstration is being run in python, so if you would like to replicate it, you will need an IDE such as VSC or Jupyter Notebook to execute it.

One library will also need to be installed for the demonstration to work.
phe 1.5.0 - documentation can be found at: https://pypi.org/project/phe/

Run the following code in either your terminal or notebook for installation:

- pip install phe (for VSC/terminal users)
- !pip install phe (for Jupyter users)

What are we trying to show and why?

The demonstration will take the two numbers entered and add them together without decrypting them. This replicates using HE to perform banking processes such as managing money in an account. It will then search through the initial inputs to see if the value 5 is present and return true or false also without decrypting the data. This replicates using HE to search for suspicious transactions or account details.
Lastly, it will return how long the whole process took to show performance metrics.

Setting up

My demonstrations steps will be shown in a Jupyter notebook format:

- **Step 1**: Import phe library for the encryption and time library to be able to keep track of how long each calculation takes.

```
from phe import paillier
import time
```

- **Step 2**: Defining the core functions needed for the demonstration to work.

```python
# Generate public and private keys
def generate_keys():
    public_key, private_key = paillier.generate_paillier_keypair()
    return public_key, private_key

# Encrypt a value
def encrypt_value(value, public_key):
    return public_key.encrypt(value)

# Decrypt a value
def decrypt_value(encrypted_value, private_key):
    return private_key.decrypt(encrypted_value)

# Perform secure addition
def secure_add(encrypted_value1, encrypted_value2, public_key):
    return encrypted_value1 + encrypted_value2
```

- **Step 3**: Bringing it all together while adding the time tracking functionality.

```python
def HE(a,b):
    # Record the start time
    start_time = time.time()

    # Generate keys
    public_key, private_key = generate_keys()

    # Encrypt values
    encrypted_value1 = encrypt_value(a, public_key)
    encrypted_value2 = encrypt_value(b, public_key)

    # Perform secure addition
    encrypted_sum = secure_add(encrypted_value1, encrypted_value2, public_key)

    # Decrypt result
    result = decrypt_value(encrypted_sum, private_key)
    print(f"Decrypted result of addition: {result}")

    # Search for value
    query_value = 5
    encrypted_query = encrypt_value(query_value, public_key)

    # Searching through encrypted values
    encrypted_values = [encrypted_value1, encrypted_value2]
    search_result = any(decrypt_value(encrypted_value, private_key) == query_value for encrypted_value in encrypted_values)

    print(f"Search result for query value {query_value}: {search_result}")

    # Record the end time
    end_time = time.time()

    # Calculate the time taken
    time_taken = end_time - start_time

    print(f"Time taken to run the cell: {time_taken} seconds")
```

Results

- Inputs 5 &10:

```
HE(5,10)
```

```
Decrypted result of addition: 15
Search result for query value 5: True
Time taken to run the cell: 2.098048210144043 seconds
```

- Inputs 3 &7000:

```
HE(3,7006)
```

```
Decrypted result of addition: 7009
Search result for query value 5: False
Time taken to run the cell: 1.5647656917572021 seconds
```

- Inputs 15497452654876254 & 1245789636985:

```
HE(15497452654876254,1245789636985)
```

```
Decrypted result of addition: 15498698444513239
Search result for query value 5: False
Time taken to run the cell: 2.7325711250305176 seconds
```

- Showing the variables the calculations are being done on:

```
print(encrypted_value1)
print(encrypted_value2)
```

```
<phe.paillier.EncryptedNumber object at 0x000001543F347CE0>
<phe.paillier.EncryptedNumber object at 0x000001543F347CB0>
```

Summary

As the results show, all calculations returned with correct values and the search function could read the encrypted data and tell us if our sought-after value (5 in this case) was present before the addition was performed. While it is a simple approach, it does showcase some of strengths of HE.

However, this demonstration does also highlight the inconsistency in the calculation time. Due to the computational complexity needed to perform the encryptions, the fact that a simple addition calculation plus search function takes more than a second is not ideal when considering how much we would need to scale up to be able to be used in a financial setting.