Housing Price Prediction using Linear Regression on input encrypted by Paillier and FHE

Vera Li, Chunxiao Wang, and Alice Yin
Team “Alice and Bobs”
05.09.22

In this presentation:

Introduction
- Problem - ML with Encrypted data
- Workflow of program

ML Model Construction
- Dataset and Preprocessing
- Model Metrics

Encryption Using Paillier
- Encryption and Decryption
- Prediction computation

Encryption Using FHE
- Encryption and Decryption
- Prediction computation

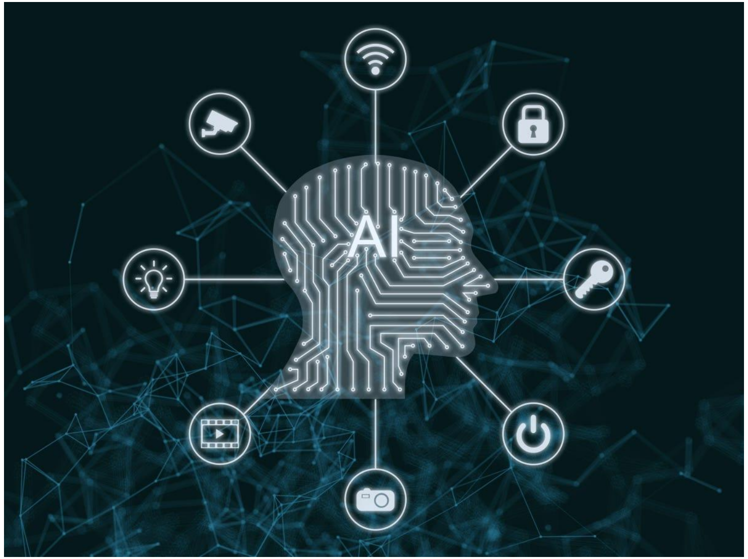
Demo Video

Results and Evaluation

Conclusion and Future Work

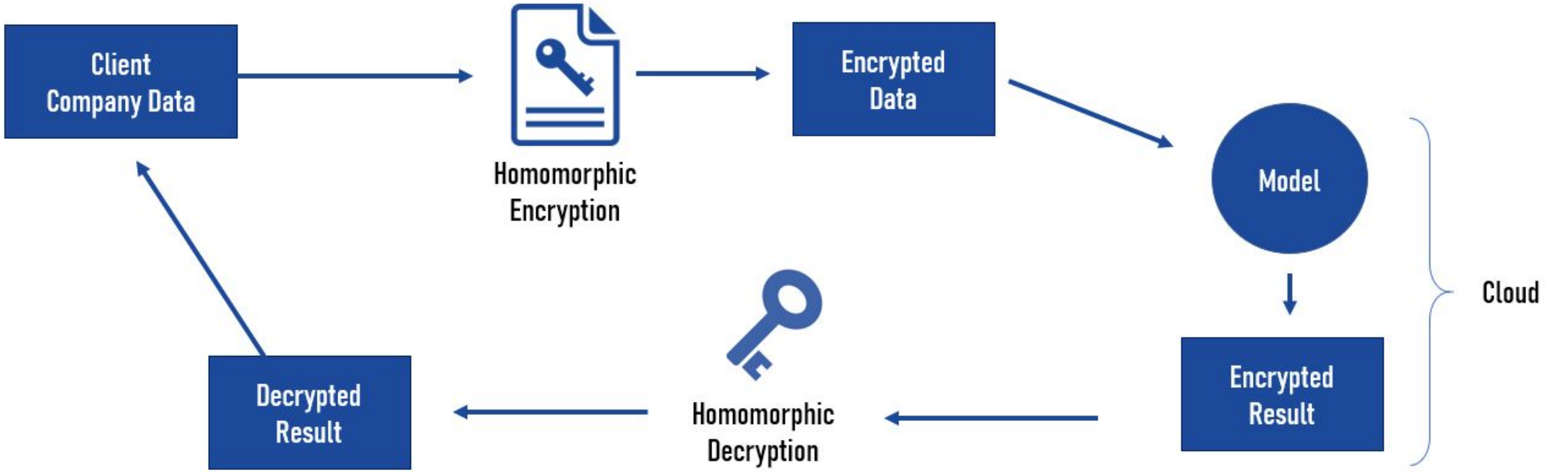
Introduction

Machine Learning with Encrypted Data



+

Workflow of Program



https://towardsdatascience.com/homomorphic-encryption-machine-learning-new-business-models-2ba6a4f185d

Machine Learning Model Construction

A Linear Regression Model

$$\begin{bmatrix} y_1 \\ y_2 \\ y_3 \\ \vdots \\ \vdots \\ \vdots \\ y_m \end{bmatrix} = \begin{bmatrix} 1 & x_{11} & x_{12} & \cdots & \cdots & x_{1n} \\ 1 & x_{21} & x_{22} & \cdots & \cdots & x_{2n} \\ 1 & x_{31} & x_{32} & \cdots & \cdots & x_{3n} \\ \vdots & \vdots & \vdots & \cdots & \cdots & \vdots \\ \vdots & \vdots & \vdots & \cdots & \cdots & \vdots \\ \vdots & \vdots & \vdots & \cdots & \cdots & \vdots \\ 1 & x_{m1} & x_{m2} & \cdots & \cdots & x_{mn} \end{bmatrix} \begin{bmatrix} \beta_0 \\ \beta_1 \\ \beta_2 \\ \beta_3 \\ \vdots \\ \vdots \\ \beta_m \end{bmatrix}$$

$$Y = X\beta$$

Dataset and Preprocessing

- California Housing Data

Lot Area	Overall Qual	Overall Cond	Total BsmtSF	Gr Liv Area	Tot Rms Abv	Garage Area	Sale Price
8450	7	5	856	1710	8	548	208500
9600	6	8	1262	1262	6	460	181500
11250	7	5	920	1786	6	608	223500
9550	7	5	756	1717	7	642	140000

- Data Standardization

$$Z = \frac{x - \mu}{\sigma}$$

https://www.kaggle.com/competitions/house-prices-advanced-regression-techniques/data

Model Metrics

- Coefficients of Model

```
lr = LinearRegressionModel()
lr.train()
lr.getCoef()
```

```
array([ 7.36004848e-02,  4.47617617e-01,  3.64404974e-02,  1.55312197e-01,
        2.83291920e-01, -3.27711529e-04,  1.55766813e-01])
```

- Recall:

$$Y = X\beta = x_1\beta_1 + x_2\beta_2 + \ldots$$

- Computation Efficiency with plaintext input:

```
--- 1464 Predictions finished in 0.09264636039733887 seconds ---
```

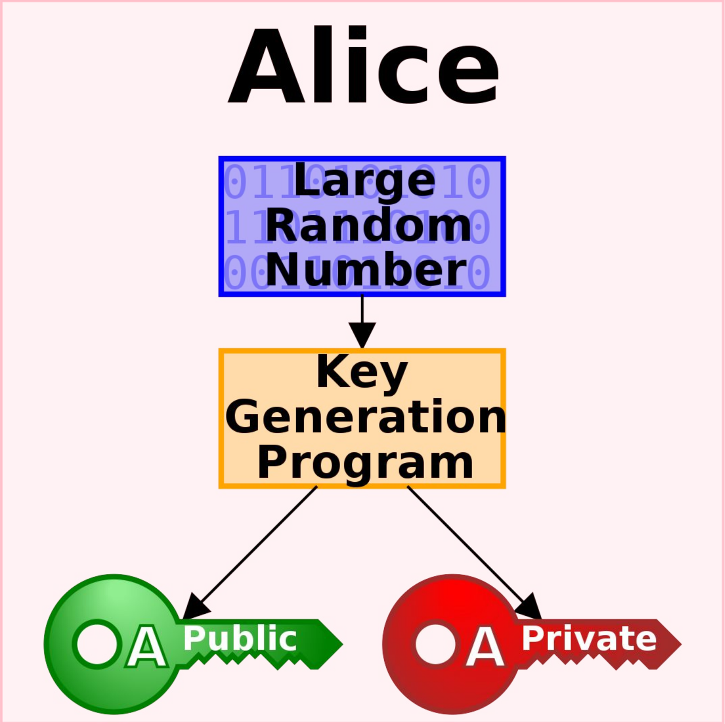
Encryption Scheme Requirements

$$Y = X\beta = x_1\beta_1 + x_2\beta_2 + \ldots$$

- Sum-homomorphism
- (Multiplication by a known value)-malleability

Encryption Using Paillier

Encryption and Decryption




```python
#Encryption using Paillier
def Paillier_encrypt(userInput):
    pub_key, priv_key = getKeys()
    temp = []
    for k,val in userInput.items():
        temp.append(int(val))
    data = lotArea, OverallQual, OverallCond, TotalBsmtSF, GrLivArea, TotRmsAbvGrd, GarageArea = temp

    #encrypt the data and generate a json file to send to the company
    encrypted_data_list = [pub_key.encrypt(x) for x in data]
    encrypted_data = {}
    encrypted_data['public key'] = {'n': pub_key.n}
    encrypted_data['values'] = [(str(x.ciphertext()), x.exponent) for x in encrypted_data_list]
    datafile = json.dumps(encrypted_data)
    with open('data.json', 'w') as file:
        json.dump(datafile, file)
```

```python
#Decrypt the data print it for customers
def Paillier_decrypt():
    pub_key, priv_key = getKeys()
    answer_file = loadAns()
    answer_key = paillier.PaillierPublicKey(n = int(answer_file['pub_key']['n']))
    answer = paillier.EncryptedNumber(answer_key, int(answer_file['values'][0]), int(answer_file['values'][1]))

    #only decrypt when the public key in answer match with our public key - to verify this is the expecting result
    if(answer_key == pub_key):
        print(priv_key.decrypt(answer))
```

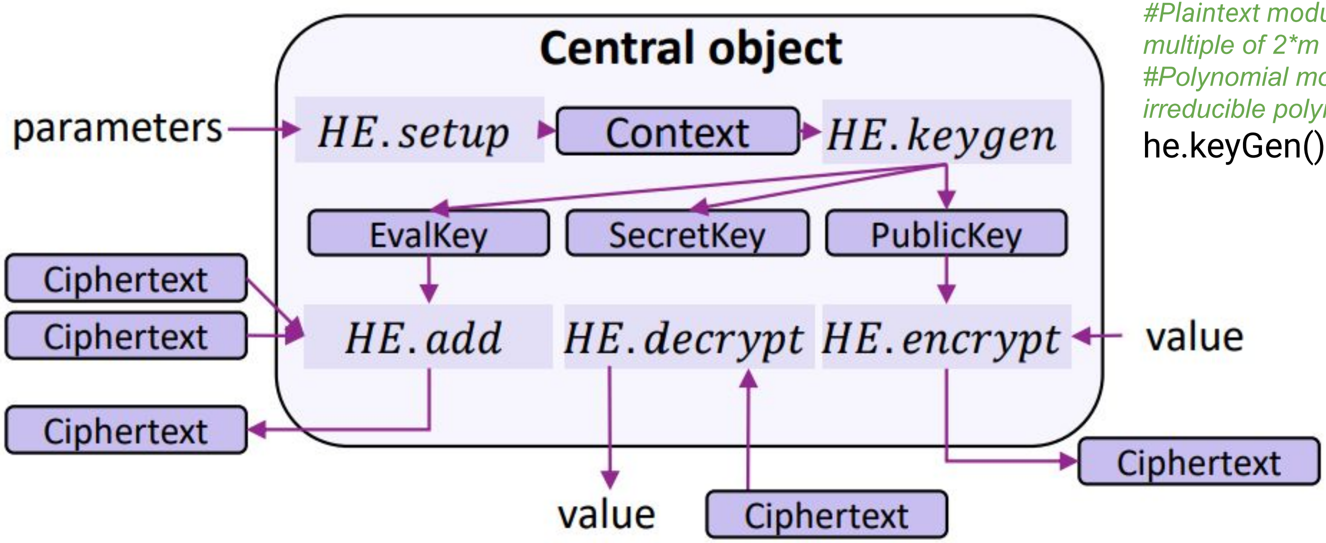
Prediction Computation

Computation of results
Multiplication and addition between ciphertext and float numbers

```
def computePaillier(price_mean,price_std, lr,mean,std):
    data = getInputPaillier()
    mycoef = lr.getCoef()
    pk = data['public key']
    pub_key = paillier.PaillierPublicKey(n = int(pk['n']))
    enc_nums_rec = [paillier.EncryptedNumber(pub_key, int(x[0], int(x[1]))) for x in data['values']]
    result = sum([mycoef[i] * normalize(enc_nums_rec[i],mean[i],std[i]) for i in range(len(mycoef))])
    result = result*price_std
    result = result + price_mean
    return result, pub_key
```

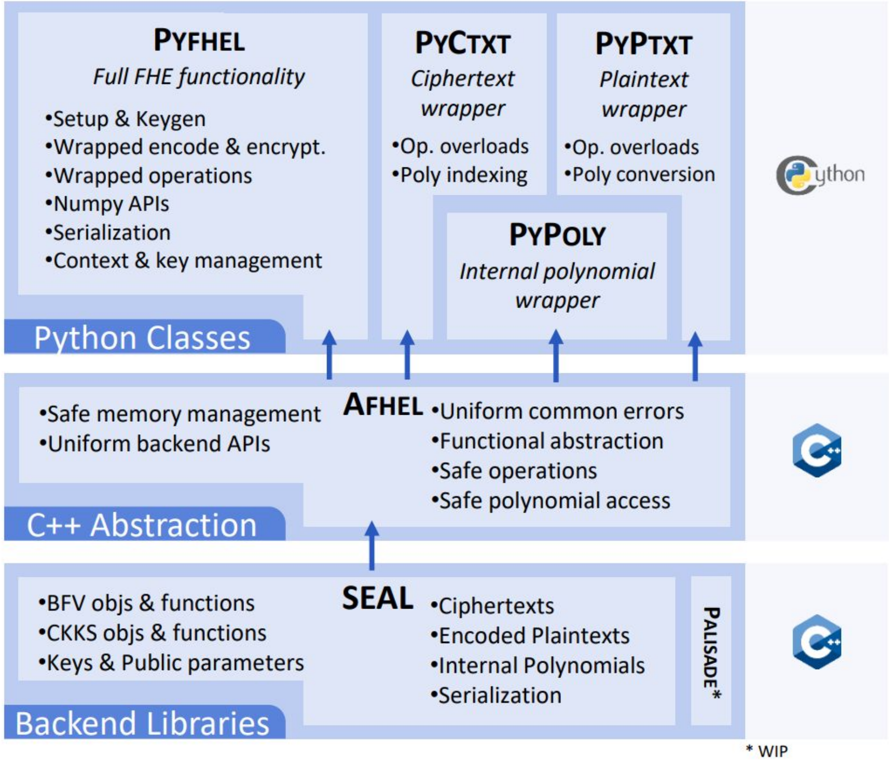
Encryption Using FHE

Design Principle




```
he = Pyfhel()
he.contextGen(p=65537, m=4096)
#Plaintext modulus (p): This is prime and p-1 is multiple of 2*m
#Polynomial modulus (m): This is the degree of the irreducible polynomial x^m+1.
he.keyGen()
```

Architecture



Python Classes

Abstraction for Homomorphic Encryption Libs for safe and uniform C++ encapsulation of different backend APIs

Backend: FHE libraries written in C++

Encryption and Decryption

```python
#Encryption scheme for FHE
def Enc(data, HE):
    f = open("cipher.p", "wb")
    data["res"] = float(0)
    for k,v in data.items():
        ptxt = HE.encodeFrac(v)
        ctxt = HE.encryptPtxt(ptxt)
        data[k] = ctxt
    data["he"] = HE
    pickle.dump(data,f)
    f.close()
```

```python
def FHE_decrypt():
    data = pickle.load(open("res.p", "rb"))
    res = data["res"]
    HE = data["he"]
    HE.restorepublicKey('pub.key')
    HE.restoresecretKey('secret.key')
    r = HE.decryptFrac(res)
    print("The result is " + str(r) + "\n")
```

Serialization (Pickle): a way to convert a data structure into a linear form that can be stored or transmitted over a network.

Prediction Computation

```
def computeFHE(num, price_mean,price_std, mean, std):
  data = pickle.load(open("cipher.p", "rb"))
  #instances imported, secret key excluded
  HE = data["he"]
  del data["he"]
  #Initialize the instances of the pyfhel ciphertext objects
  for k,v in data.items():
    v._pyfhel = HE
  res = data["res"]
  del data["res"]
  i = 0
  for k,v in data.items():
    v = normalize(v,mean[i],std[i])
    mul = v * num[i]
    res = res + mul
    i+=1
  r = res*price_std + price_mean
  #store the result in a file to return back to the client side
  output = dict()
  output["res"] = r
  output["he"] = HE
  pickle.dump(output, open("res.p", "wb"))
```

Computation of results
Multiplication and addition between
PyCtxt Objects and float numbers

Program Demo Video

Results and Evaluation

PAILLIER

	1	2	3	4	5
Encryption(s)	13.30	35.81	48.74	70.37	84.54
Computation(s)	1.77	3.66	5.25	7.73	9.81
Decryption(s)	0.71	1.41	2.12	2.82	3.54
Total(s)	15.78	40.88	56.11	80.92	97.89

PYFHEL

	1	2	3	4	5
Encryption(s)	0.33	0.64	0.94	1.27	1.61
Computation(s)	0.34	0.68	1.06	1.39	1.73
Decryption(s)	0.22	0.44	0.66	0.9	1.16
Total(s)	0.89	1.76	2.66	3.56	4.5

Conclusion and Future Work

We have demonstrated that...

- The computation results for both encryption schemes are consistent with the plaintext computation result.
- Paillier generally takes longer time than FHE (Pyfhel)

We can potentially improve the program by…

- Responsive Server
- Server client file space separation
- Classifier

Thank you!