Applied Cryptography and Trust

Coursework Submission for CSN11131

Edinburgh Napier University 2020-2021 Trimester 2

Enabling Machine Learning using Homomorphic Encryption

Name – Devi Patnaik

Matric No – 40502606

Keywords: Cryptography, Homomorphic Encryption, Machine Learning, Data Protection

Abstract: With the increasing use of online resources like banking, social media, voting, surveys, medical trials etc., it becomes very important that the privacy of the individuals' data is preserved. More so, when there is a very high uptake on machine learning and its utilisation in prediction of events and results. This work presents an evaluation of how homomorphic encryption and machine learning can come together to preserve the sanctity of data.

1. Introduction

Organisations carry out their business in a connected world. They implement Enterprise-wide software packages like Enterprise Resource Planning, Customer Relationship Management software and so on. These are software packages that link the business process across the various functional areas of an organisation. Sometimes, these organisations link software platforms across to partner organisations. Companies link their systems to their customers and vendors through EDI links. Various parts of the business use business intelligence and machine learning to make informed decisions related to the business. And most of this, nowadays is cloud based.

At an individual level, we exchange data with various platforms like banks, social media, government agencies, e-commerce platforms, medical institutions, and a number of cloud-based services. All this is done in return for a particular privilege. All these agencies provide some vital services.

It becomes obvious that the one thing that travels around the organisation is 'data'. Raw data moves along the system, gets transformed, manipulated, operations are applied to it. And it provides the output. All the service providers assure the users that data in-transit and at rest. The vulnerability is when the data is in its raw form as a plaintext in memory.

Within the organisation, one can have it in its raw form. Sometimes and more often currently, with the advent of web-based computing, secure channels have come into play. There is still the man in the middle who can break in and tap into useful data.

With this backdrop, this paper will study the concept of homomorphic encryption and test if a method can be developed where in different operations can be performed on data in its cyphertext form while it is still in the memory so that only the sender and receiver are able to decipher what the output of data in its raw form.

The study will examine a few libraries and some test programs to establish the efficacy of such a possibility. Simple statistical operations will be tested on a small scale to evaluate this concept.

Mathematical and statistical operations will be carried out on data in plain text and then replicated with the cyphertext. We use Python implementations of Pailler scheme to carry out these tests. Examples are provided so that the concept is useful to audience who would like to implement homomorphic encryption is their machine learning solutions.

The paper will aim to provide a literature review of some relevant observations on this topic.

2. Literature Review and Design

The personally identifiable information (PII) as it is known comes with a lot of concerns. Users of various platforms and devices have worries about identities being stolen, tampered with and above all used for malicious activities.

Data moved from being stored in in-premise servers to cloud-based servers. As such people had very little faith in in-premise servers. Now data is going to some server in a remote server farm. Moreover, with the privacy laws surrounding GDPR being tightened, it becomes even more important that personal information is not compromised with.

The major challenge was to carry out mathematical operations like additions and multiplications.

Rivest, Adleman and Dertouzos [1] worked on the homomorphic encryption. At the same time Rivest, Shamir and Adleman [2] were working on the RSA which supported multiplications. This was further enhanced by ElGamal [3].

Paillier (1999) [4] and before him Goldwasser and Micali [5] worked on the additions. However, it was not possible to do simultaneous additions and multiplications till Gentry [6] in his work demonstrated fully homomorphic schemes. Fan and Vercauteren [7] popularly known as FandV did more solid work on large polynomials.

Computationally, homomorphic encryption has constraints in the sense that small amounts of data could potentially end up requiring large amount of space for their cypher text size. But proposals of mitigating this situation have been put forward by works of Naehrig et.al., [8]

All this leads us to explore how to enable machine learning. Statistical packages like R and Python support homomorphic calculations with the Phe.Paillier and homomorphR libraries. Microsoft AI laboratory has provided the Simple Encrypted Arithmetic Library (SEAL) [9] enabling end-to-end encrypted data storage.

Partial homomorphic encryption allows a limited set of operations like addition and multiplication.

Encrypt(a) + Encrypt(b) = Encrypt(a+b)

Encrypt(a) * Encrypt(b) = Encrypt(a*b)

Somewhat homomorphic encryption allows operations like multiplication on polynomials. For example., for Encrypt(x1)…Encrypt(x2) we can compute Encrypt(p(x1….xn)) for a polynomial p(x1…xn) with a low degree bound.

There are some issues with this. One the size of the cypher text and two the errors or noise as it is called.

Gentry tackled this with bootstrapping techniques. Brakerski and Vaikuntanathan came up with concepts of learning with errors and combined it with bootstrapping to enhance it further [14].

Fully homomorphic encryption allows for additions and multiplications with 'AND', 'OR', and 'XOR' operations.

These platforms enable machine learning models in cloud to take encrypted data, compute and produce the encrypted output which can only be decrypted by the holder of the private key.

Lattice based encryption is the latest technology which is being employed in achieving fully homomorphic encryption.

The Microsoft SEAL library allows the cloud service to perform the computations while it protects the customer's data using the cryptographic algorithms. The cloud gets to consume encrypted data and result of computation is revealed to the customers. This is an open-source library for expanding the research and utilisation of homomorphic encryption.

IBM has done ground-breaking work in fully homomorphic encryption based on bootstrappable encryption scheme. This enables efficient scheme for bilinear forms. Distributed computing and efficient two-party oblivious transfer is carried out efficiently as well. There is a lot of research being done in Lattice Cryptography by IBM.

3. Implementation

As mentioned in the introduction, organisations and individuals have a need to consume data and provide it as well. Individuals and organisations provide vital data to service providers and the service providers return the favour by providing the resultant output.

Many organisations take their transactional data to the next level and feed data to machine learning algorithms. Historic data is used to train the algorithm and then the test case is provided to the algorithm to predict future outcomes. The resource hungry nature of these computations does not lend it to be a premise-based solution. More and more organisations are now looking to cloud based machine learning and artificial intelligence platforms to do the number crunching and provide the resultant outcome.

Data leaves the individual or the organisation and gets consumed by the cloud-based platform. It is this scenario where the homomorphic encryption will come into play. Data will be sent in an encrypted form. Computations will be carried out over this encrypted data and the output will be provided in a form that the receiver of data can be sure that it has not be compromised with.

In this section a simple exercise is carried out where a python program calculates the mean of 100 integers. A similar exercise is also carried out on the encrypted numbers. The

mean will be calculated on the encrypted numbers and compared to the calculations done to the numbers in plain text.

In the first instance, a simple python program does a loop of 100 numbers as below:

HEExample3.py

```
import sys
import  random
from phe import paillier

x =0
y =0
z=0

print("Normal Calculations")
mylist = list(range(1,101))
x= sum(mylist)
y= len(mylist)
print ("Sum: ",x)
print ("Len: ",y)
z= x/(y-1)
print ("Avge: ",z)

print("Homomorphic Calculations")
pub, priv = paillier.generate_paillier_keypair()
a = pub.encrypt(x)
b = pub.encrypt(1/y)
print ("Encrypted Sum:",a)
print ("Encrypyted inverse of Len:" ,b)
c=priv.decrypt(b)
d = a * c
print("Multiplication of Encrypted Value a with scalar c", d)
e=priv.decrypt(d)
print ("Decrypted Avge:", e)
```

As can be seen, this is a very simple program in python. But it demonstrates that mathematical operations like addition and multiplication are supported by homomorphic encryption.

This calculation can be part of a cloud machine learning platform. The data can be submitted to the engine in encrypted format and the output could be decrypted in a plaintext format, exactly how the user would have seen had they done the calculation in plaintext.

The second example is carried out on the R Studio platform used extensively for statistical computations and driving machine learning models. The homomorphR library was used to encrypt and decrypt values and compare that with the normal mathematical computation of addition. Here again the both the methods of computation gave identical results as shown in the RExampleHE.R program below.

RExampleHE.R program

```
library (homomorpheR)
keyPair<- PaillierKeyPair$new(modulusBits=1024)

a<-gmp::as.bigz(2173849)
#Normal Addition
d<- (a+10)
d
#Homomorphic Addition
b<- keyPair$pubkey$encrypt(a+10)
#Decryption of the result
c<- keyPair$getPrivateKey()$decrypt((b))
#Compare Normal Computation to Homomorphic Computation
identical(d,c)
```

Results

```
> library (homomorpheR)
> keyPair<- PaillierKeyPair$new(modulusBits=1024)
>
> a<-gmp::as.bigz(2173849)
> #Normal Addition
> d<- (a+10)
> d
Big Integer ('bigz') :
[1] 2173859
> #Homomorphic Addition
> b<- keyPair$pubkey$encrypt(a+10)
> #Decryption of the result
> c<- keyPair$getPrivateKey()$decrypt((b))
> #Compare Normal Computation to Homomorphic Computation
> identical(d,c)
[1] TRUE
```

One of the major concerns with data is personally identifiable information could be misused. Also, rendering to a lot of crossover research, pharmaceutical companies lack trust in each other. It is in this area data can be exchanged using homomorphic encryption and machine leaning models could do the computations on encrypted data and provide the results. Certain columns can be encrypted in a data set and the data privacy maintained.

Harrison, E. et al [12] discuss the use cases of various kinds of applications like blinding trials, recontacting participants, long-term follow up of participants etc.by storing pseudoanonymised format.

Smart wearable devices like Fitbit, apple watch etc., are loaded with health applications. Users exchange data with cloud-based servers to store data. This is one area the usage of homomorphic encryption is most required. The device's lightweight memory and storage is a constraint, and any solution needs to take that into account.

Pandemics like Covid have created situations where social distancing is of utmost importance. Crowds of voters at voting centres serve as super spreaders. It is this area, where online voting and secure identity and polling counts could be implemented by applying homomorphic encryption to voters personally identifiable information.

Corporate organisations are moving to cloud based payroll processing firms due to their inability to keep up with government legislation. Specialist cloud-based payroll applications are processing payroll. Data sent to these services could be encrypted so that key information like NI numbers, date of birth etc can be encrypted using homomorphic encryption.

4. Evaluation

4.1 Python Paillier

The first example in Section 3, generates a random public and private key pair based on the Python-Paillier public Github repository [10]. This scheme works only on integer values. An EncryptedNumber object gets instantiated, which contains the cyphertext. This EncryptedNumber can then be used for the mathematical operations like addition, subtraction, and multiplication. The multiplication can only be done with a scalar value, although the result is still preserved as an encrypted value. This result can be decrypted with the private key generated during the key generation process.

The entire operation can be explained as in the Figure 1 below.

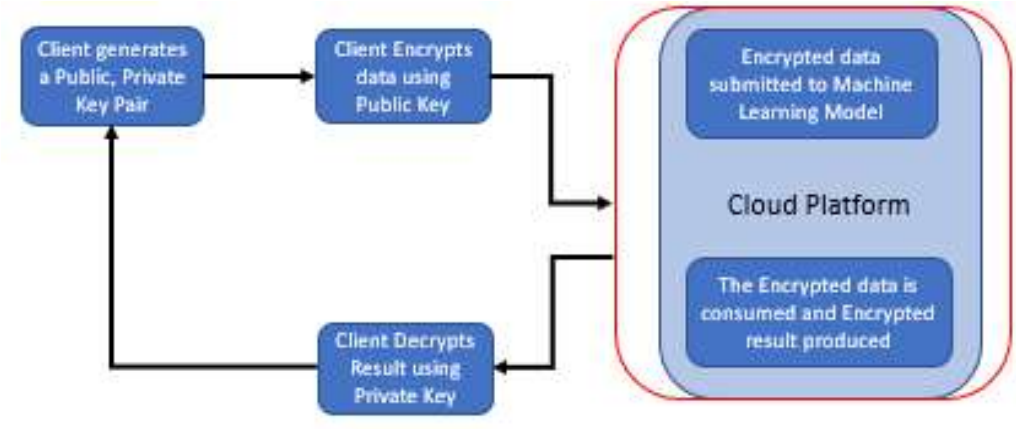

Figure 1. Flow of calculations in Homomorphic Encryption.

4.2 R Studio

The package used to demonstrate the homomorphic encryption in this paper is homomorpheR which implements the Paillier system [11]. This is demonstrated in the 2nd example in Section 3. This provides partial homomorphism, especially additive. The notation for this can be shown as below in Figure 2 below.

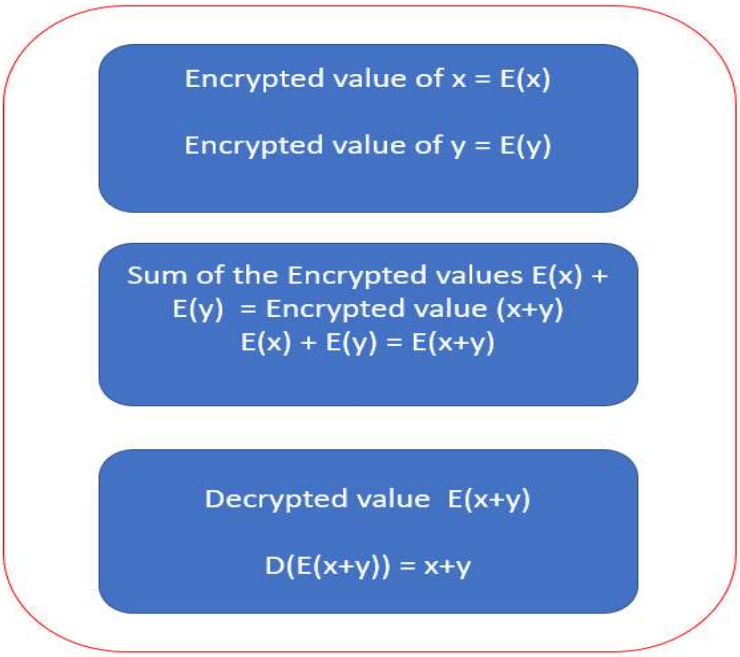

Figure 2. Addition in homomorphR

4.3 Computational Constraints

Aslett, Louis et. al., [13] discuss about a few constraints while evaluating homomorphic encryption. The message gets inflated substantially after encryption. An integer of 4 bytes could result in cypher text the size of 65,536 bytes based on the scheme presented in Section 2.3 of their paper. That means 1MB data set will end up occupying 16.4 GB of encrypted data.

The speed of calculations is also a constraint. HomomorphicEncryption implemented in R package achieves thousands of additions per second but only about 50 multiplications per second. Full CPU parallelism could be an answer to mitigate this situation.

At the moment no homomorphic scheme supports divisions and comparisons. A lot of work needs in development of these features.

Due to the randomness in cyphertext encryption there is a constraint in the depth at which mathematical operations can take place.

5. Conclusion

We can see that there is a lot of areas where homomorphic encryption can be used to enable secure machine learning models. Vital parts of user data can be encrypted and provided to the machine learning models in cloud-based environments safe in the knowledge that personally identifiable information will be protected.

A lot of work has been carried out since the initial days of research into encryption. There is still a lot that needs to be done to remove the constraints of computational abilities so that a combination of all mathematical operations can be done without putting systems under the strain of speed of computation, space etc. With storage space becoming cheaper by the day, the space issue could probably be mitigated.

Developments in areas like quantum computing etc augurs well for homomorphic encryption as the speed of computation is bound to increase as well.

The work being done by leaders like IBM and Microsoft to enhance and provide laboratories and open-source libraries to develop this field of homomorphic encryption could eventually help overcome the constraints faced and pave way for a fully homomorphic encryption enabling fool proof security in machine learning models.

6. References

[1] Rivest, R. L., Adleman, L. and Dertouzos, M. L. (1978), 'On data banks and privacy homomorphisms', Foundations of Secure Computation 4(11), 169–180.

[2] Rivest, R. L., Shamir, A. and Adleman, L. (1978), 'A method for obtaining digital signatures and public-key cryptosystems', Communications of the ACM 21 (2), 120-126.

[3] ElGamal, T. (1985), A public key cryptosystem and a signature scheme based on discrete logarithms, in 'Advances in Cryptology', Springer, pp. 10–18.

[4] Paillier, P. (1999), Public-key cryptosystems based on composite degree residuosity classes, in 'Advances in Cryptology - EUROCRYPT'99', Springer, pp. 223–238.

[5] Goldwasser, S. and Micali, S. (1982), Probabilistic encryption & how to play mental poker keeping secret all partial information, in 'Proceedings of the fourteenth annual ACM symposium on Theory of computing', ACM, pp. 365–377.

[6] Gentry, C. (2009), A fully homomorphic encryption scheme, PhD thesis, Stanford University. URL: crypto.stanford.edu/craig

[7] Fan, J. and Vercauteren, F. (2012), 'Somewhat practical fully homomorphic encryption', IACR Cryptology ePrint Archive.

[8] Naehrig, M., Lauter, K. and Vaikuntanathan, V. (2011), Can homomorphic encryption be practical?, in 'Proceedings of the 3rd ACM workshop on Cloud computing security workshop', ACM, pp. 113–124.

[9] Microsoft SEAL - GitHub - microsoft/SEAL: Microsoft SEAL is an easy-to-use and powerful homomorphic encryption library. (Web Source)

[10] Python-Paillier – Github Repository https://github.com/data61/python-paillier (Web Source)

[11] Narasimham, B. (2019), Introduction to Homomorphic Computation in R. https://cran.r-project.org/web/packages/homomorpheR/vignettes/introduction.html (Web Source)

[12] Harrison, E., Pius, R. (2021), R for Health Data Science https://argoshare.is.ed.ac.uk/healthyr_book/ (Web Source)

[13] Aslett, Louis J. M., P. Esperança and Chris C. Holmes. "A review of homomorphic encryption and software tools for encrypted statistical machine learning." ArXiv abs/1508.06574 (2015): n. pag.

[14] Brakerski, Z., Vaikuntnathan, V., (2011) Efficient Fully Homomorphic Encryption from (Standard) LWE, Foundations of Computer Science. 1975, 16th Annual Symposium on 2011 (2); 97-106