Calculating the average using Paillier's cryptosystem

Christiana Zaraket
ESIB, USJ CIMTI
Beirut, Lebanon
christiana.zaraket@net.usj.edu.lb

Maroun Chamoun
ESIB, USJ CIMTI
Beirut, Lebanon
maroun.chamoun@usj.edu.lb

Tony Nicolas
ESIB, USJ CIMTI
Beirut, Lebanon
tony.nicolas@usj.edu.lb

***Abstract*—Homomorphic encryption (HE) is one of the efficient ways that allow many companies to store their data in an encrypted form into the Cloud and then the latter can be analyzed and worked with as if it were still in its initial form. In this paper we consider two types of homomorphic encryption (HE) schemes: the additive and the multiplicative ones. But we will be focusing on the additive one in order to calculate on the cipher-texts and get the average while decrypting the result. Therefore we will show the theoretical part of the additive homomorphic Paillier's cryptosystem which is used in order to encrypt and decrypt the messages needed, but what can be more interesting is showing in details the application part of it which is realized by a numerical example applied on Sagemath.**

Keywords—Homomorphic encryption (HE), additive homomorphic, Paillier's cryptosystem

I. INTRODUCTION

Cloud computing is a data storing technique that gives opportunities for out-sourcing of storage and computation. It offers flexibility and cost saving, but the main disadvantage that faces many companies to use the Cloud computing in their work is the security concept. The first solution proposed was to encrypt the sensible data before storing it into the Cloud, but if ever we want to apply operations on the latter without decrypting it and without having any information about the initial data, we need to use what's called homomorphic encryption (HE). The homomorphic encryption (HE) procedure is simple: the user stores its encrypted data into the cloud and sends encrypted queries over it, the latter must be able to send back to user an answer which is in an encrypted form and while decrypting the result, the user obtains what he wants. In addition a homomorphic encryption (HE) scheme could be either additive such as Paillier's [1] and Goldwasser-micali's [2] cryptosystems or multiplicative such as RSA [3] and EL Gamal [4] crypto-systems. This depends on what the user wants as operation to be done on his plain-texts after applying the decryption function. After all these work, in 2009 Craig Gentry had the idea of creating a new cryptosystem known as Fully homomorphic encryption scheme (FHE) [5] which can handle addition and multiplication operations at the same time, but such scheme was proved unfeasible in practice due to a massive overhead in computation and memory cost. However, the idea of this article comes from the murex's purpose, which is to calculate a list of KPI's such as calculating the average salary for the 10% higher salary and the average monthly earnings... After studying the elements of this list, we found that the operations needed by the latter are the addition operation and the dividing by a constant operation and then we found a new simple technique to calculate the average of given plain-texts by only using their appropriate cipher-texts.

This paper is the result of a prolonged study on Paillier's cryptosystem and our purpose of the latter is to facilitate and to make clearer the mathematical ideas and theorems used in this cryptosystem. All this is done by giving details and proofs on every step taken, as well by giving numerical examples. So that a reader, from any field, can understand not only the Paillier's cryptosystem but also can see direct applications on it.

The rest of this paper is decomposed as follows: in section II we define the notion of homomorphic encryption (HE) as well as its properties. In section III, we present in details Paillier's cryptosystem which is an example of a homomorphic encryption (HE) scheme and we show, in section IV, how to apply operations on it. In section V, we give a numerical example concerning the previous section. And finally we conclude this paper in section VI by giving a conclusion and an idea about future work.

II. HOMOMORPHIC ENCRYPTION

In this section we give a definition of a homomorphic encryption (HE) for an asymmetric scheme as well as its properties.

A. Asymmetric Scheme [6]

An asymmetric scheme is based on three functions:

- KeyGen( ):generates two different keys, a public one noted pk and a secret one noted sk.
- Encrypt(pk, m_i) : is a mathematical function that is able to transform a plain-text m_i into a cipher-text c_i using the public key pk.
- Decrypt(sk, c_i): is a mathematical function that uses the secret key sk in order to obtain the plain-text m_i from the cipher-text c_i.

B. Homomorphic Encryption (HE) for an Asymmetric Scheme

An asymmetric homomorphic encryption scheme, schematized in the Fig. 1, is realized by following these steps:

As a first step the client must encrypt his messages m_i using Encrypt(pk, m_i) function, and then he sends the results to be stored into the Cloud. The next step could be that the client sends some queries f( ) to the Cloud that must be capable to compute an encryption of $f(m_i)$ noted by Encrypt(pk,$f(m_i)$) and we mention here that f $\in \{+, \times\}$. Then the Cloud returns this result to the client who will compute Decrypt(sk,Encrypt(pk,$f(m_i)$)) and obtains $f(m_i)$.

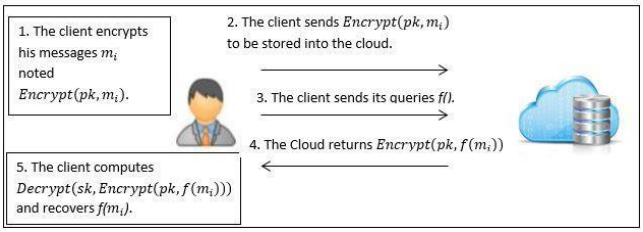

Fig. 1 A client-Cloud homomorphic encryption (HE) scenario

All this process, schematized in the Fig. 1, is done without having to decrypt the cipher-texts stored into the Cloud and without having any information on the initial data.

C. *Homomorphic Properties*

There are two types of homomorphic encryption:

- An encryption scheme is supposed to be additive homomorphic if it verifies the following:

$$g(Encrypt(pk, m_i))=Encrypt(pk, \sum_{i=1}^{n} m_i).$$

- An encryption scheme is supposed to be multiplicative homomorphic if it verifies the following:

$$g(Encrypt(pk, m_i))=Encrypt(pk, \prod_{i=1}^{n} m_i).$$

We mention here that g $\in \{+, \times\}$.

III. PAILLIER'S CRYPTOSYSTEM [1]

Paillier's cryptosystem is an example of additive homomorphic encryption scheme invented by Pascal Paillier in 1999. We give in this section an explanation of the Paillier's cryptosystem construction and its properties.

A. *Scheme Construction*

Let n be the multiplication of two chosen prime numbers p and q and let g be an element of $Z^*_{n^2}$. But in the rest of this article we will consider that g is equal to n+1 as Damgard, Jurik and Nielsen has chosen it in 2010 in order to make the choice of g the simplest possible [7].

The public key is given by pk=(n,g) and the secret one is given by sk=(p,q).

The key generation algorithm using Sagemath is the following:

```
def keygen(KeySize):
    p=random_prime(2**( KeySize //2),lbound=2**(
    KeySize //2-1),proof=True)
    q=random_prime(2**( KeySize //2),lbound=2**(
    KeySize //2-1),proof=True)
    n=p*q
    g=n+1
    lmd=(p-1)*(q-1)
    mu=(lmd**(-1))%n
    return p,q,n,g,lmd,mu
```

To encrypt a plain-text m $\in Z_n$ one can use the following encryption function:

$$c=Encrypt(pk, m)=g^m r^n \bmod n^2,$$

where r is a random element from Z^*_n.

This can be represented on Sagemath as following:

```
def encrypt(m,n,g):
    r=random_prime(n,proof=True)
    return ((g**m)*(r**n))% n**2
```

The decryption function is given by:

$$Decrypt(sk, c)=\frac{L(c^\lambda \bmod n^2)}{L(g^\lambda \bmod n^2)} \bmod n$$

Where $L(u)=\frac{u-1}{n}$ and the Carmichael's function λ is equal to $lcm(p-1,q-1)$.

On Sagemath this function could be represented as following:

```
def decrypt(c,lmd,mu,n):
    return ((((c**lmd)%(n**2)-1)/n)*mu)%n
```

Remark: The decryption function gives back the plain-text m because:

$$\begin{aligned} c^\lambda \bmod n^2 &= (g^m r^n)^\lambda \bmod n^2 \\ &= g^{m\lambda} r^{n\lambda} \bmod n^2 && (1) \\ &= g^{m\lambda} \bmod n^2 && (2) \\ &= (1+n)^{m\lambda} \bmod n^2 && (3) \\ &= \sum_{i=0}^{m\lambda} C_i^{m\lambda} n^i \bmod n^2 && (4) \\ &= (C_0^{m\lambda}.1 + C_1^{m\lambda}.n + C_2^{m\lambda}.n^2 + \\ &\quad n^2(\sum_{i=3}^{m\lambda} C_i^{m\lambda} n^{i-2})) \bmod n^2 && (5) \\ &= (1+nm\lambda) \bmod n^2 && (6) \end{aligned}$$

The second part of the decryption function which is $\mu = (L(g^\lambda \bmod n^2))^{-1}$, is calculated in the key generation algorithm and so that of λ in order to not being obliged to repeat the same calculation each time we are using the Decryption algorithm and therefore we are optimizing the calculation time of the latter. μ is calculated as follows:

$$\begin{aligned} g^\lambda \bmod n^2 &= (1+n)^\lambda \bmod n^2 \\ &= (1+n\lambda) \bmod n^2 \end{aligned}$$

$$\begin{aligned} L(g^\lambda \bmod n^2) &= \frac{1+n\lambda-1}{n} \bmod n \\ &= \lambda \bmod n \end{aligned}$$

So $\mu = \lambda^{-1} \bmod n$

And then,

$$Decrypt(sk, c)=\frac{L(c^\lambda \bmod n^2)}{L(g^\lambda \bmod n^2)} \bmod n = \frac{\frac{1+nm\lambda-1}{n}}{\lambda} \bmod n = m \bmod n$$

Before passing to the next section, we would like to explain a little bit about the equalities we have considered while calulating $c^\lambda \bmod n^2$.

- To pass from (1) to (2) we have used the Carmichael's theorem [8], which is the following:

For every element $w \in Z^*_{n^2}$

$$\begin{cases} w^\lambda = 1 \bmod n \\ w^{n\lambda} = 1 \bmod n^2 \end{cases}$$

In our case we have $r \in Z^*_n$ which is equivalent to $r \in Z^*_{n^2}$ and $0<r<n$,

and therefore while applying Carmichael's theorem one can get $r^{n\lambda}=1 \bmod n^2$.

- To pass from (2) to (3) we have replaced g by (1+n).
- To pass from (3) to (4) we have used the binomial theorem [9] which is the following:

$$(x+y)^n = \sum_{k=0}^{n} C_k^n x^{n-k} y^k$$

Where the n choose k combinations formula is equal to:

$$C_k^n = \frac{n!}{k!(n-k)!}$$

- To pass from (4) to (5), we have to develop the formula $\sum_{i=0}^{m\lambda} C_i^{m\lambda} n^i \bmod n^2$.
- To pass from (5) to (6), we calculate each term of the equation (5) as follows:
  - $C_0^{m\lambda}.1=\frac{(m\lambda)!}{0!(m\lambda-0)!}.1=1$
  - $C_1^{m\lambda}.n=\frac{(m\lambda)!}{1!(m\lambda-1)!}.n= m\lambda n$
  - $(C_2^{m\lambda}.n^2+n^2(\sum_{i=3}^{m\lambda} C_i^{m\lambda} n^{i-2})) \bmod n^2 =0$ because $(\text{integer}\times n^2) \bmod n^2=0$

B. Paillier's Cryptosystem Properties

Suppose that we have two plain-texts m_1, m_2, with two cipher-texts respectively Encrypt(pk,m_1), Encrypt(pk,m_2).

Paillier's cryptosystem verifies the following equations:

- Decrypt(Encrypt(pk,m_1)Encrypt(pk,m_2) $\bmod n^2$)=(m_1+m_2) mod n [1]. (7)
- Decrypt(Encrypt(pk,m_1)^k $\bmod n^2$) = k. m_1 mod n [10]. (8)

Equation (7) shows that Paillier's cryptosystem is additive homomorphic.

IV. Applying Operations On Paillier's Cryptosystem

Calculating on the cipher-texts and getting the average while decrypting the results could be interesting.

To realize that, we need to follow four steps:

- Step 1: Calculate an encryption of $\sum_{i=1}^{k} m_i \bmod n$ by using (7) which means by calculating $\prod_{i=1}^{k} c_i \bmod n^2$ and we consider here that k is the number of elements needed. This step is applied on Sagemath as following:

```
def encrypted_sum(M,n,g):
    C=[encrypt(m,n,g) for m in M ]
    return mul(C) % (n**2)
```

- Step 2: Calculate an encryption of $k^{-1} \sum_{i=1}^{k} m_i \bmod n$ by using (8) which means by calculating $[\mu] = (\prod_{i=1}^{k} c_i \bmod n^2)^{k^{-1} \bmod n} \bmod n^2$. This step is applied on Sagemath as following:

```
def encrypted_mean(M, n,g):
    k=(len(M)**(-1))% n
    return (encrypted_sum(M,n,g)**k)%(n**2)
```

- Step 3: Apply Decrypt( ) function on $[\mu]$ to obtain $\frac{\sum_{i=1}^{k} m_i}{k} \bmod n$ which is the average but in Z_n.

Proof:

Decrypt(sk,$[\mu]$)

$=\text{Decrypt}(sk,(\prod_{i=1}^{k} c_i \bmod n^2)^{k^{-1} \bmod n} \bmod n^2)$

$=\text{Decrypt}(sk, (\text{Encrypt}(pk, \sum_{i=1}^{k} m_i \bmod n))^{k^{-1} \bmod n} \bmod n^2)$

$=(k^{-1} \bmod n . \sum^{k} m_i$

$=(k^{-1} \sum_{i=1}^{k} m_i \bmod n) \bmod n$ (using (8))

$)\bmod n.$

This step is applied on Sagemath as following:

```
def dec_mean(M,lmd,mu,n,g):
    return decrypt(encrypted_mean(M,n,g),lmd,mu,n)
```

- Step 4: Applying LLL algorithm or the Extended Euclidean Algorithm to obtain the average result in Q.

Proof:

1) LLL Algorithm idea
   - The LLL algorithm purpose

     Let a_1, a_2 be two vectors that form a basis of a lattice L. The goal of LLL algorithm is to take as input a_1, a_2 and to give as output a new basis for the lattice L where the lengths of the vectors of the latter are as short as possible.

     One of the most important properties of the LLL algorithm is that the first vector given as output represents the shortest one in the lattice L [11].

   - The use of LLL algorithm in our case

     Suppose that μ^* is equal to $\frac{\sum_{i=1}^{k} m_i}{k}$ such that gcd($\sum_{i=1}^{k} m_i$,k) and gcd(k,n) are equal to 1. And suppose that μ represents the latter in Z_n so that it could be written in this way: $\frac{\sum_{i=1}^{k} m_i}{k} \bmod n$. The condition of having gcd(k,n)=1, taken before assure the existence of k^{-1} in μ.

     In our case, this form is obtained after applying Decrypt formula on $[\mu]$. So we know the value of μ and not that of $\sum_{i=1}^{k} m_i$ nor that of k. A way to find the values of the latters is to apply the two-dimensional lattice theory.

     We define a lattice L as following:

     $L = \{(x,y) \in Z^2; x = y\mu \bmod n\}$ (9)

     which is equivalent to:

     $L = \{(x,y) \in Z^2; kx = y \sum_{i=1}^{k} m_i \bmod n\}$ (10)

     From (9), one can say that (n,0) and (μ,1)

form a basis of L which is correct for two reasons:

- First, because one can notice that $n=0\mu \bmod n$ therefore $(n,0) \in L$ and one can also notice that $\mu = 1\mu \bmod n$ therefore $(\mu,1)\in L$.
- Second, we have $n\times 1-0\times\mu\neq 0$ therefore $(n,0)$ and $(\mu,1)$ are not collinear.

From (10), replacing x by $\sum_{i=1}^{k} m_i$ and y by k one can deduce that $(\sum_{i=1}^{k} m_i, k)$ is a vector of L. So obtaining the values of $\sum_{i=1}^{k} m_i$ and k that verifies $\gcd(\sum_{i=1}^{k} m_i, k)=1$, means that the fraction $\frac{\sum_{i=1}^{k} m_i}{k}$ is irrational and then the latters' values are optimal. This is equivalent to finding the shortest vector of the lattice L that can be obtained by the application of the LLL algorithm on the basis vectors $(n,0)$ and $(\mu,1)$ [12].

- LLL algorithm using Sagemath:

```
def lattice_LLL(N,avg):
    M=Matrix(2,2,[avg,1,N,0]) #The vectors of the basis in a matrix form
    L=M.LLL() #The new basis vectors
    return L[0,0]/ L[0,1]
    #Return the fraction formed by the shortest vector
    #elements which corresponds to
    ∑_{i=1}^{k} m_i / k .
```

- Average Function using LLL algorithm in Sagemath:

```
def avg_LLL(M,lmd,mu,n,g):
    return lattice_LLL(n,dec_mean(M,lmd,mu,n,g))
```

2) Extended Euclidean Algorithm idea [11]

The Extended Euclidean Algorithm gives as result the shortest vector in a given two-dimensional lattice L. The only thing that differs the Extended Euclidean Algorithm from LLL Algorithm is that the latter can handle higher dimensions. But in our case we just need two dimensional lattices therefore the two algorithms can be applied and give the same result.

To improve a given basis the Extended Euclidean Algorithm follows these steps:

a) Subtract from a vector a linear combination of the others which means that in each time the initial vector length is getting shorter.
b) Swap two vectors.

- Extended Euclidean Algorithm using Sagemath

```
def ExtendedEuclide(N,avg):
    u1=0
    u2=N
    v1=1
    v2=moy
    while u2>sqrt(N):
        Q = u2 // v2
        [t1,t2]=[u1-v1*Q,u2-v2*Q]
        [u1,u2]=[v1,v2]
        [v1,v2]=[t1,t2]
    return u2/u1
```

- Average Function using Extended Euclidean algorithm in Sagemath:

```
def avg_Euclide(M,lmd,mu,n,g):
    return ExtendedEuclide (n,dec_mean(M,lmd,mu,n,g))
```

V. Practical Example Of Calculating The Average Using The Cipher-Texts

Using Sagemath, we show in this part an example that illustrates the previous sections.

```
[nb, min, max, KeySize] = [10, 100, 500, 24]
random.seed(int(time.time()))
M=[random.randint(min,max) for m in range(nb)]
p,q,n,g,lmd,mu=keygen(KeySize)
print("Average using LLL= "+str(avg_LLL(M,lmd,mu,n,g)))
print("Average using Extended Euclide= "+str(avg_Euclide(M,lmd,mu,n,g)))
```

Suppose that we have p = 3623 and q = 3833. In this case we have n=pq= 13886959, g=n+1= 13886960, λ= 13879504 and $[L(g^{\lambda} \bmod n^2)]^{-1}$= 594224.

Using the public key pk=(n,g), the secret key sk=(p,q) and random numbers r, one can compute cipher-texts given by $c=\text{Encrypt}(pk, m)=g^m r^n \bmod n^2$ as shown in the table below.

Plain-texts	441	140	461	245	371	437	414	420	473	284
r	13178449	1401761	229519	6847147	3308609	42349	6965261	5544613	11394839	188833
Cipher-texts	12813770471784	9063629260735	4406234026281	9429975964247	3430363106355	1170528073 04693	7287385245 9248	114251522214695	7958338598 2690	4015382474 4941

In our case, the average of the plain-texts is equal to 1843/5. To obtain it using only the cipher-texts one must follow the steps given in section IV.

As mentioned before the first step is to calculate $[\mu] = (\prod_{i=1}^{k} c_i^{2^{k-1} \bmod n} \bmod n) \bmod n$ which is equal in our case to 401021692061012. In the second step, while decrypting the latter; one can obtain 8332544 as a result. Finally, to have

the average value in Q, one must apply the Extended Euclidean Algorithm or the LLL algorithm and obtain 1843/5. So this example make us sure that in this way we can always calculate safely on the cipher-texts and obtain the same result as if we are applying the average on the plain-texts.

VI. Conclusion And Future Work

In this paper, we reviewed the companies' problem concerning the security issue while using the Cloud Computing in their work and how it can be solved by the homomorphic encryption idea. We present in particular the Murex's KPIs list based on addition and ordering operations. For additional issue we used an important additive homomorphic scheme known by Paillier's cryptosystem and we reviewed a numerical application on it. However, in fact, we realized that this cryptosystem could not achieve the ordering idea because of the existence of the modulus in its encryption function.

Based on this problem, future work will focus on finding a new secured cryptosystem that can handle the additive and the ordering issue at the same time. Our objective is to allow Murex Company (and other companies that provide technology solutions to financial market) to calculate their KPIs list easily and efficiently.

Acknowledgment

We thank Dr. Eric Filiol (ESIEA University, France) for valuable discussions and feedback.

We would also like to thank Murex Company for placing their trust and confidence in our abilities to achieve our purpose.

Finally, the authors would like to acknowledge the National Council for Scientific Research of Lebanon (CNRS – L) for granting a doctoral fellowship for Christiana Zaraket.

References

[1] Naveed ISLAM, William PUECH and Robert BROUZET, How to Secretly Share the Treasure Map of the Captain? Proc. SPIE 7542, Multimedia on Mobile Devices 2010, 75420L (27 January 2010); https://doi.org/10.1117/12.839844

[2] Shruthi R, Sumana P, Anjan K Koundinya, Performance Analysis of Goldwasser-Micali Cryptosystem, International Journal of Advanced Research in Computer and Communication Engineering, Vol. 2, Issue 7, July 2013

[3] Nikita Somani, Dharmendra Mangal, An Improved RSA Cryptographic System, International Journal of Computer Applications (0975 – 8887), Volume 105 – No. 16, November 2014.

[4] Andreas V. Meier, The ElGamal Cryptosystem, June 8, 2005.

[5] Craig Gentry.2009. Fully homomorphic encryption using ideal lattices. In Proceedings of the forty-first annual ACM symposium on Theory of Computing (STOC'09). ACM, New York, NY, USA, 169178:https://doi.org/10.1145/1536414.1536440

[6] Khalil Hariss, Maroun Chamoun and Abed Ellatif Samhat, On DGHV and BGV Fully Homomorphic Encryption Schemes, 1st Cyber Security in Networking Conference (CSNet), Rio de Janeiro, Brazil, 18-20 Oct. 2017.

[7] Nina Pettersen, Applications of Paillier's Cryptosystem, NTNU, Norwegian University of Science and Technology, August 2016.

[8] Andreas Steffen, The Paillier Cryptosystem, Hochschule für Technik Rapperswil, 17.12.2010, Paillier.pptx 1.

[9] Brett Berry, The Binomial Theorem Explained wi8th a special splash of Pascal's Triangle, Oct 26, 2018.

[10] Mohamed Nassar, Abdelkarim Erradi, Qutaibah M. Malluhi, Paillier's Encryption: Implementation and Cloud Applications. International Conference on Applied Research in Computer Science and Engineering (ICAR)

[11] Phong Nguyễn, The LLL Algorithm, May 2010, Luminy, http://www.di.ens.fr/~pnguyen.

[12] Pierre-Alain Fouque, Jacques Stern, and Geert-Jan Wackers, CryptoComputing with rationals. In Financial Cryptography, volume 2357 of Lecture Notes in Computer Science, pages 136-146. Springer, Berlin, Heidelberg 2002.