Sistema criptográfico Paillier

O criptossistema de Paillier é um criptossistema assimétrico inventado por Pascal Paillier em 1999. A sua segurança baseia-se na DCRA (*decisional composite residuosity assumption*).

DCRA é uma suposição matemática que declara que dado um número composto *n* e um inteiro *z*, é computacionalmente dificíl de decidir se *z* é um *n*-ésimo resíduo de módulo n^2, isto é:

```
z≡y^n (mod n^2)
```

Este esquema é homomórfico, isto é, apenas com acesso à chave pública e à encriptação de *m1* e *m2*, é possível executar a encriptação de *m1+m2*. Este sistema está provado que é seguro contra um invasor passivo.

Algoritmo

1.Geração de chaves

O par de chaves é gerado da seguinte forma: são gerados dois números primos aleatórios *p* e *q* e calcula-se *n* que é a multiplicação entre os dois números, de forma que o máximo divisor comum de *n* e (*p*-1)(*q*-1) seja 1. Esta propriedade é assegurada se os dois números primos sejam do mesmo tamanho, para isso utiliza-se 1024 bits.
De seguida calcula-se o λ que é o mínimo multiplo comum de (p-1)(q-1).
O anel de inteiros é o conjunto de inteiros construído sobre uma estrutura algébrica Z. Neste caso define *Zn2* como o anel de n^2 (IntegerModRing(n^2)), selecionando um inteiro *g* que pertença a esse conjunto, que é *n*+1.

```
def keygen():
    bits=1024
    p, q = random_prime(2^bits), random_prime(2^bits)
    n = p*q
    print("n",n)
    lmda = lcm(p-1,q-1)
    print ("lambda",lmda)
    Zn2 = IntegerModRing(n^2)
    g = Zn2(n+1)
    return (n,lmda,g)
```

2.Encriptação

Neste passo da encriptação, recebe-se a mensagem *m* e os inteiros *n* e *g*.
Define-se um número aleatório *r* pertencente ao mesmo anel de *g*.
Concluindo, calcula-se a cifra *c*, sendo que $c = g^m \cdot r^n \pmod{n^2}$

```
def encrypt(m,n,g):
    Zn2 = IntegerModRing(n^2)
    r = Zn2(randint(1, n))
    c = g^m * r^n % n^2
    return c
```

3.Desencriptação

No passo da desencriptação, recebe-se *c*, que é o resultado da encriptação da mensagem, o *lambda* e os inteiros *n* e *g*.
Para se obter o valor da mensagem que inicialmente se pretendia encriptar, faz-se a seguinte operação:

$$m = \frac{L(c^{\lambda} \mod n^2)}{L(g^{\lambda} \mod n^2)} \mod n$$

L(x) é a função *(x-1)/n*
Como se está a operar com elementos do anel de inteiros, utiliza-se a função lift no caso de *x.lift()*

```
def decrypt(c,lmda,n,g):
    up =L((c^lmda) % (n^2),n)
    down = L((g^lmda) % (n^2),n)
    m = (up // down) % n
    return m
```

```
def L(x,n):
    r = (x.lift()-1) / n
    return r
```

Define-se a mensagem *m* a encriptar. Esta mensagem tem que pertencer ao conjunto pré-definido, isto é, $0 \leq m < n$. Neste exemplo vamos supor que *m* tem o valor de 64.
Depois de termos gerado as chaves, de ter encriptado a mensagem, e de ter o resultado final da desencriptação, procedemos à comparação desse valor com a mensagem inicial. Se os os resultados forem iguais significa que o sistema funcionou.

```
n,lmda,g = keygen()
m=64
c = encrypt(m,n,g)
print("c",c)
d = decrypt(c,lmda,n,g)
print("Mensagem inicial igual ao resultado da desencriptacao?",m==d)
```

```
('n',
995424067210530755225707556937465116115003938938581843679782413797306020135775120410999997447335975568349106680696815399880156767631484023194536941458686672714687199563710484747112372110968473605225707156278334919990024118027976554899838554095694277472971485760726006739272796180065011831998991541007059898387061224957239135943875062476243298922175166288720840699491219098173749521898569186842837802793973804322739996317515313639709802800995175399600530744310453584582362223201574177413985364407384015913679089021926554897069941700376504118178188730971053796283144619588053403532790801589099491121128278682154986777 3)
('lambda',
497712033605265377612853778468732558057501969469290921839891206898653010067887560205499998723667987784174553340348407699940078383815742011597268470729343336357343599781855242373556186055484236802612853578

139167459995012059013988277449919277047847138736485742880363003369636398090032505915999495770503529949183209397993071998425292650280440847398307595300704068863735783272433139514810069793756774213944771327058152925107377633150077302138566501206810081915739060432342885923755532510510679224458666132089241602042018333775511424474225611975439891390784685423499117570464470001630306969930579891582896592744547589 9390)
```

```
('c',
791487512397576891560426958860536889159106191932459686107918368875098923086872025441819611467537098362763659963845001625278945643283075804876939075726963101796224894952269227321562900283270974527766865894424776100191356981410270874745270965942507038495301819390879002509197615038619997445318305198965271623729790779554894605502237929877837543394102124395715973575282632106570736359985222606841320321185791775298309571881390888973352272596240205309188136903074412484658523186986215641341669024046586609037409148334791527688110553450042285910542623354570289502635588421892430083835356916083879706723439246593054381342476315421308230707075899351687628045060622376256467291981192324115646481347287539611759142078246186170822740813890619205289108880336785603552923958867307782823911872619542899700982268551397627039071357165833701229602453857687095683773462506832554715081813900753115091998794005498699789971393524008682596564133782369600578640572056109184389536887871031705872516579619154776709528786724818618360514999433278347578668719668786391402048261424650043376394428515435383503937050396891127139105983248079663254093226610168554225967481025285943363322210635832146847761277800263108330260701581187706001832419628868391642987022 79)
('Mensagem igual ao resultado da desencriptacao?', True)
```