TP3 - Ex.1

Abril 25, 2024

Estruturas Criptográficas

PG53886, Ivo Miguel Alves Ribeiro

A95323, Henrique Ribeiro Fernandes

Hidden Number Problem

Parâmetros segundo Neguyen & Schparlinski

- Um primo $p \approx 2^d$
- O número de bits mais significativos $k > \sqrt{d} + \lvert d \rvert)$
- A dimensão do reticulado $n > 2\sqrt{d}$

```
from sage.all import*

def k_NS(p):
    return ceil(sqrt(log(p,2))) + ceil(log(log(p,2),2))

def n_NS(p):
    return 2*ceil(sqrt(log(p,2)))

class HNP(object):
    def __init__(self,p=257):

        self.n = n_NS(p)
        self.red = None

        if not p.is_prime():
            raise ValueError("Não é primo ",p)
        self.p = p

        k = k_NS(p)
        if 2**k >= p:
            raise ValueError("Bits a mais",k)
        self.lam = 2**k

    ##extraçao dos bits mais significativos
    def msb(self,y):
        p = self.p; L = self.lam
        return floor(y*L/p)
```

```
    def problema(self ,s=None ,x=None ,u=None):
        # Parâmetros
        p = self.p;

        s = ZZ.random_element(1,p)
        ### criaçao de um lista x random
        self.xs = [ZZ.random_element(1,p) for i in range(self.n)]
        ### criaçao da lista u calculada apartir do segredo e do x
        self.us = [self.msb(s*x % p) for x in self.xs ]
        self.s = s

    def reducao(self,Big=None):
        # parâmetros
        p = self.p; L = self.lam ; B = p/L ; A = 1/L
        if not Big:
            Big = L*p  ## qualquer inteiro grande
        n   = self.n
        m   = 2 + n

        # Construção das matrizes
        linha1 = Matrix(QQ,1,m,[x for x in self.xs]+[A,0])
        linha2 = Matrix(QQ,1,m,[-u*B for u in self.us]+[0,Big])
        czeros = Matrix(QQ,n,1,[0]*n)
        qident = p*identity_matrix(ZZ,n)
        linha3 = block_matrix(QQ,1,3,[qident,czeros,czeros])
        G      = block_matrix(QQ,3,1,[linha3,linha1,linha2])

        # Redução LLL
        self.G = G.LLL()

    ### testa se o segredo calculado é igual ao segredo original
    def teste(self):
        L = self.lam ; p = self.p

        erro   = self.G[-1][-2]
        s_calc = floor(erro * L) % p

        if self.s == s_calc:
            print ("\nsegredo calculado ")
            return s_calc
        else:
            raise ValueError("\nvalor incorreto do segredo")


d = 512
p = random_prime(2^d, lbound = 2^(d-1))
```

Teste

```
hnp = HNP(p=p)
hnp.problema()
hnp.reducao()
hnp.teste()


segredo calculado

76853471275504069400117254214366971743712808819321281083904960131360382806389825947135053661322338345332163769705550301905202287901490407857110395938332 07
```