EX1_TP4

May 27, 2024

[86]:
```
import hashlib
from random import randint
from sage.all import *
```

[87]:
```
n = 256
q = 8380417
h = 60
r = 1753
k = 4
l = 3
gamma = 523776
alpha = 261888
eta = 6
beta = 325

# FUNÇÕES AUXILIARES
def gerar_S(limite, tamanho):
    Zq.<z> = PolynomialRing(GF(q))
    Rq.<z> = Zq.quotient(z^n + 1)
    S = []
    for _ in range(tamanho):
        pol = [randint(1, limite) for _ in range(n)]
        S.append(Rq(pol))
    S = matrix(Rq, tamanho, 1, S)
    return S

def decompor(c, t):
    Rq = IntegerModRing(q)
    Rt = IntegerModRing(t)
    r = int(Rq(c))
    r0 = int(Rt(r))
    if r0 > t / 2:
        r0 -= int(t)
    if r - r0 == q - 1:
        r1 = 0
        r0 -= 1
    else:
        r1 = (r - r0) // int(t)
    return (r1, r0)

def obter_bits_altos(c):
    x = decompor(c, 2 * alpha)
    return x[0]

def obter_bits_baixos(c):
    x = decompor(c, 2 * alpha)
    return x[1]

def extrair_bits_altos(pol):
    k = pol.list()
    for i in range(len(k)):
        h = k[i].list()
        for j in range(len(h)):
            h[j] = obter_bits_altos(int(h[j]))
        k[i] = h
    return k

def extrair_bits_baixos(pol):
    k = pol.list()
    for i in range(len(k)):
        h = k[i].list()
        for j in range(len(h)):
            h[j] = obter_bits_baixos(int(h[j]))
        k[i] = h
    return k

def amostra_na_bola(r):
    sl = "{:064b}".format(int(r[:8].hex(), 16))
    k = 8
    c = [0] * 256
    for i in range(196, 256):
        while int(r[k]) > i:
            k += 1
        j = int(r[k])
        k += 1
        s = int(sl[i - 196])
        c[i] = c[j]
        c[j] = (-1) ** s
    return c

def shake(a, b, n=256):
    shake = hashlib.shake_256()
    shake.update(a)
    shake.update(b)
    return shake.digest(n)
```

```
def funcao_hash(a, b, n=256):
    r = shake(a, b, n)
    return amostra_na_bola(r)

def norma_infinita(pol):
    return max(max(abs(int(coeff)) for coeff in poly.list()) for poly in pol.
 ↪list())

def norma_infinita_vetor(vetor):
    return max(norma_infinita(vetor[i]) for i in range(vetor.nrows()))

def norma_infinita_matriz(matriz):
    return max(max(abs(el) for el in row) for row in matriz)
```

[88]:
```
#FUNÇÕES PRINCIPAIS
def gerar_chaves():
    Zx.<x> = ZZ[]
    R.<x> = Zx.quotient(x^n + 1)
    Zq.<z> = PolynomialRing(GF(q))
    Rq.<z> = Zq.quotient(z^n + 1)

    K = []
    for i in range(k * l):
        K.append(Rq.random_element())
    A = matrix(Rq, k, l, K)

    s1 = gerar_S(eta, l)
    s2 = gerar_S(eta, k)

    t = A * s1 + s2

    chave_publica = (A, t)
    chave_privada = (s1, s2)
    return chave_publica, chave_privada

def assinar_mensagem(m, chave_publica, chave_privada):
    A = chave_publica[0]
    s1 = chave_privada[0]
    s2 = chave_privada[1]

    Zx.<x> = ZZ[]
    R.<x> = Zx.quotient(x^n + 1)
    Zq.<z> = PolynomialRing(GF(q))
    Rq.<z> = Zq.quotient(z^n + 1)

    z = None
    while z is None:
        y = gerar_S(gamma - 1, l)
        Ay = A * y
        w = extrair_bits_altos(Ay)

        u = str(w).encode()
        k = m.encode()
        c = funcao_hash(k, u)
        cq = Rq(c)

        z = matrix(y + cq * s1)
        if norma_infinita_vetor(z) >= (gamma - beta) or␣
 ↪norma_infinita_matriz(extrair_bits_baixos(Ay - cq * s2)) >= (alpha - beta):
            z = None

    return (z, c)

def verificar_assinatura(m, assinatura, chave_publica):
    Zx.<x> = ZZ[]
    R.<x> = Zx.quotient(x^n + 1)
    Zq.<z> = PolynomialRing(GF(q))
    Rq.<z> = Zq.quotient(z^n + 1)

    A = chave_publica[0]
    t = chave_publica[1]
    z = assinatura[0]
    c = assinatura[1]
    cq = Rq(c)

    Az = A * z
    w = extrair_bits_altos(Az - cq * t)

    u = str(w).encode()
    k = m.encode()
    novo_c = funcao_hash(k, u)

    if norma_infinita_vetor(z) >= (gamma - beta) or c != novo_c:
        print('Assinatura rejeitada!')
    else:
        print('Assinatura aceita!')
```

[89]:
```
chave_publica, chave_privada = gerar_chaves()
m = "Hello World"
assinatura = assinar_mensagem(m, chave_publica, chave_privada)
verificar_assinatura(m, assinatura, chave_publica)
```

```
Assinatura aceita!
```

```
[ ]:
```