qTesla

gen ¶

Geração de uma chave privada e uma chave pública para assinatura de uma mensagem. Em A selecionam-se k polinómios aleatórios do anel Rq. Em s seleciona-se um polinómio segundo uma distribuição Gaussiana e em e selecionam-se k polinómios segundo uma distribuição Gaussiana. Calcula-se t com os valores anteriores. Geram-se, respetivamente, a chave privada e a pública:

- sk = (s, e, A)
- pk = (A, t)

sign

Função que, recebendo uma mensagem e uma chave privada, faz a assinatura da mensagem, retornando esta assinatura no final. Para tal necessita da chave e de um polinómio aleatório do anel Rq (y). No final aplica uma função de *hash* sobre A1*y..Ak*y. Calcula-se também z, y+s*c1..y+s*ck. Retorna (z, c).

verify

Recebendo uma mensagem, uma assinatura e uma chave privada, verifica que a chave corresponde (ou não) à mensagem passada como argumento. Deverá retornar True caso tal se verifique e False caso contrário.

Imports

In [272]:

```
import os
from sage.stats.distributions.discrete_gaussian_polynomial import DiscreteGaussianDistr
ibutionPolynomialSampler
from hashlib import shake_128
```

Variáveis globais

In [273]:

```
n = 256
q = 8380417
k = 3
sigma = 3

Gq.<z>  = PolynomialRing(GF(q))
Rq.<z> = Gq.quotient(z^n+1)
```

Funções

In [274]:

```
def gen():
    A = [Rq.random_element() for _ in range(k)]
    s = DiscreteGaussianDistributionPolynomialSampler(Rq, 64, sigma)()
    e = [DiscreteGaussianDistributionPolynomialSampler(Rq, 64, sigma)() for _ in range(
k)]

    t = [A[i] * s + e[i] for i in range(k)]

    sk = (s, e, A)
    pk = (A, t)

    return (sk, pk)


def sign(m, sk):
    (s, e, A) = sk
    y = Rq.random_element()
    #print([A[i]*y for i in range(k)])
    c = [hash(A[i]*y) for i in range(k)] + [hash(m)]
    z = [y + s*c[i] for i in range(k)]
    return (z,c)

def verify(m, pk, sig):
    (z, c) = sig
    (A, t) = pk
    w = [A[i]*z[i]-t[i]*c[i] for i in range(k)]
    #print('------------------------------------------------------------------------
------------------------')
    #print(w)
    return c == [hash(w[i]) for i in range(k)] + [hash(m)]
```

Teste

Devido a um erro de implementação, a verificação falha.

In [275]:

```
m = Rq.random_element()
(sk,pk) = gen()
(z,c) = sign(m, sk)
verify(m, pk, (z,c))
```

Out[275]:

```
False
```

In [ ]: