Dilithium

June 7, 2021

1 TP3 - Estruturas Criptográficas

1.1 Elementos do grupo 4:

- André Morais, A83899
- Tiago Magalhães, A84485

2 DILITHIUM

2.0.1 Criação dos parâmetros

[38]:
```
import os
import math
import sys
import hashlib
import numpy as np

class DILIT:
    #Geração dos parâmetros (Security Level 5)
    def __init__(self):
        self.n = 256
        self.q = 2^23 - 2^13 + 1
        self.k = 8
        self.l = 7
        self.eta = 2
        self.tau = 60
        self.beta = 120
        self.gama1 = 2^19
        self.gama2 = (self.q)-1/32

    #Geração dos anéis
        Zx.<x> = ZZ[]
        Zq.<z> = PolynomialRing(GF(self.q))
        self.Rq = QuotientRing(Zq,z^self.n+1)
        self.R = QuotientRing(Zx, x^self.n+1)

    #Geração do espaço matrix
        self.Mr  = MatrixSpace(self.Rq,self.k,self.l)
```

2.0.2 Criação das chaves

[39]:
```
class DILIT(DILIT):

    #Algoritmo de geração de chaves
    def keygen(self):
        # Criação da matriz A
        A = self.genA()

        # Criação dos vetores s1 e s1
        s1 = self.genS(self.eta, self.l)
        s2 = self.genS(self.eta, self.k)
        t = A*s1 + s2

        pubkey = (A,t)
        privkey = (A,t,s1,s2)

        return pubkey, privkey

    #Geração a matriz A em Rq
    def genA(self):
        K = []
        for i in range(self.k*self.l):
            K.append(self.Rq.random_element())
        A = self.Mr(K)
        return A

    #Geração Vetores S em Rq com o tamanho 'tam' e coeficiente até 'limite'
    def genS(self, limite, tam):
        vetor = MatrixSpace(self.Rq,tam,1)
        K = []
        for i in range(tam):
            poli = []
            for j in range(self.n):
                poli.append(randint(1,limite))
            K.append(self.Rq(poli))
        S = vetor(K)
        return S
```

2.0.3 Sign

[68]:
```
class DILIT(DILIT):
    def sign(self, sk, M):
        A, t, s1, s2 = sk
        vetor = MatrixSpace(self.Rq,self.k,1)
        z = 0
        while(z==0):
```

```
            # Criação do vetor y
            y = self.genS(int(self.gama1-1) , self.l)
            # Cálculo w
            w = A* y
            # Aplicar HighBits
            w1 = self.hbPoli(w, 2*self.gama2)
            w1_pack = str(w1).encode()
            k = M.encode()

            # Hash de M || w1
            c = self.Hash(k, w1_pack)
            cq = self.Rq(c)

            # Cálculo de z
            z = y + cq*s1

            if self.norma_inf_vet(z)[0]>= self.gama1 - self.beta and self.norma_inf_matriz(self.lbPoli(A*y-cq*s2,2*self.gama2))>= self.gama2-self.beta:
                sigma = (z,c)
                return sigma
            else:
                z=0

    def highBits(self, r, alfa):
        (r1,r0) = self.decompose(r, alfa)
        return r1

    def lowBits(self, r, alfa):
        (r1,r0) = self.decompose(r, alfa)
        return r0

    def decompose(self, r, alfa):
        r = mod(r, self.q)
        r0 = int(mod(r,int(alfa)))
        if (r-r0 == self.q-1):
            r1 = 0
            r0 = r0-1
        else:
            r1 = (r-r0)/int(alfa)
        return (r1,r0)

    def hbPoli(self, poli,alfa):
        k = poli.list()
        for i in range(len(k)):
            h = k[i]
            h = h.list()
            for j in range(len(h)):
```

```
                h[j]=self.highBits(int(h[j]), alfa)
            k[i]=h
        return k

    def lbPoli(self,poli,alfa):
        k = poli.list()
        for i in range(len(k)):
            h = k[i]
            h = h.list()
            for j in range(len(h)):
                h[j] = self.lowBits(int(h[j]),alfa)
            k[i] = h
        return k

    def access_bit(self, data, num):                              #Passa de Bytes para bits
        base = int(num // 8)
        shift = int(num % 8)
        return (data[base] & (1<<shift)) >> shift

    def SampleInBall(self,r):
        sl = [self.access_bit(r[:8],i) for i in range(len(r[:8])*8)]
        k = 8 # descartar primeiros 8
        c = [0] * 256

        for i in range (256-self.tau, 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(self,a,b):
        shake = hashlib.shake_256()
        shake.update(a)
        shake.update(b)
        s = shake.digest(int(256))
        return s

    def Hash(self,a,b):
        r = self.Shake(a,b)
        c = self.SampleInBall(r)
```

```
        return c

    def norma_infinito(self,pol,n):
        J = pol.list()
        for i in range(len(J)):
            k = J[i]
            K = k.list()
            for j in range(len(K)):
                K[j] = abs(int(K[j]))
            J[i] = K
        L = []
        for i in range(len(J)):
            L.append(max(J[i]))
        return max(L)

    def norma_inf_vet(self,vetor):
        for i in range(vetor.nrows()):
            norm = self.norma_infinito(vetor[i],self.q)
            vetor[i] = norm
        return max(vetor)


    def norma_inf_matriz(self,matriz):
        L = []
        for i in range(len(matriz)):
            k = matriz[i]
            for j in range(len(k)):
                if k[j] < 0:
                    k[j] = abs(k[j])
                L.append(max(k))
        for i in range(len(L)):
            J = []
            J.append(max(L))
        return J[0]
```

2.0.4 Verify

[69]:
```
class DILIT(DILIT):

    # Função para verificar uma assinatura
    def verify(self,pk, M, sigma):
        A,t = pk
        z,c = sigma

        cq = self.Rq(c)
```

```python
        w1 = self.hbPoli(A*z - cq*t, 2*self.gama2)

        u = str(w1).encode()
        k = M.encode()
        c_ = self.Hash(k,u)

        if c_ == c:
            print("Assinatura Válida!")
        else:
            print("Assinatura Inválida!")
```

[70]:
```python
d = DILIT()

msg = 'hello'
pk,sk = d.keygen()
sigma = d.sign(sk,msg)
d.verify(pk, msg, sigma)
```

```
R: 203 > 201
R: 253 > 206
R: 211 > 210
R: 225 > 210
R: 251 > 216
R: 236 > 218
R: 236 > 218
R: 241 > 219
R: 252 > 226
R: 242 > 237
R: 203 > 201
R: 253 > 206
R: 211 > 210
R: 225 > 210
R: 251 > 216
R: 236 > 218
R: 236 > 218
R: 241 > 219
R: 252 > 226
R: 242 > 237
R: 203 > 201
R: 253 > 206
R: 211 > 210
R: 225 > 210
R: 251 > 216
R: 236 > 218
R: 236 > 218
R: 241 > 219
R: 252 > 226
```

```
R: 242 > 237
R: 203 > 201
R: 253 > 206
R: 211 > 210
R: 225 > 210
R: 251 > 216
R: 236 > 218
R: 236 > 218
R: 241 > 219
R: 252 > 226
R: 242 > 237
R: 203 > 201
R: 253 > 206
R: 211 > 210
R: 225 > 210
R: 251 > 216
R: 236 > 218
R: 236 > 218
R: 241 > 219
R: 252 > 226
R: 242 > 237
R: 203 > 201
R: 253 > 206
R: 211 > 210
R: 225 > 210
R: 251 > 216
R: 236 > 218
R: 236 > 218
R: 241 > 219
R: 252 > 226
R: 242 > 237
R: 203 > 201
R: 253 > 206
R: 211 > 210
R: 225 > 210
R: 251 > 216
R: 236 > 218
R: 236 > 218
R: 241 > 219
R: 252 > 226
R: 242 > 237
R: 203 > 201
R: 253 > 206
R: 211 > 210
R: 225 > 210
R: 251 > 216
R: 236 > 218
R: 236 > 218
```

```
R: 241 > 219
R: 252 > 226
R: 242 > 237
R: 203 > 201
R: 253 > 206
R: 211 > 210
R: 225 > 210
R: 251 > 216
R: 236 > 218
R: 236 > 218
R: 241 > 219
R: 252 > 226
R: 242 > 237
R: 203 > 201
R: 253 > 206
R: 211 > 210
R: 225 > 210
R: 251 > 216
R: 236 > 218
R: 236 > 218
R: 241 > 219
R: 252 > 226
R: 242 > 237
R: 203 > 201
R: 253 > 206
R: 211 > 210
R: 225 > 210
R: 251 > 216
R: 236 > 218
R: 236 > 218
R: 241 > 219
R: 252 > 226
R: 242 > 237
R: 203 > 201
R: 253 > 206
R: 211 > 210
R: 225 > 210
R: 251 > 216
R: 236 > 218
R: 236 > 218
R: 241 > 219
R: 252 > 226
R: 242 > 237
R: 203 > 201
R: 253 > 206
R: 211 > 210
R: 225 > 210
R: 251 > 216
```

```
R: 236 > 218
R: 236 > 218
R: 241 > 219
R: 252 > 226
R: 242 > 237
R: 203 > 201
R: 253 > 206
R: 211 > 210
R: 225 > 210
R: 251 > 216
R: 236 > 218
R: 236 > 218
R: 241 > 219
R: 252 > 226
R: 242 > 237
R: 203 > 201
R: 253 > 206
R: 211 > 210
R: 225 > 210
R: 251 > 216
R: 236 > 218
R: 236 > 218
R: 241 > 219
R: 252 > 226
R: 242 > 237
R: 203 > 201
R: 253 > 206
R: 211 > 210
R: 225 > 210
R: 251 > 216
R: 236 > 218
R: 236 > 218
R: 241 > 219
R: 252 > 226
R: 242 > 237
R: 203 > 201
R: 253 > 206
R: 211 > 210
R: 225 > 210
R: 251 > 216
R: 236 > 218
R: 236 > 218
R: 241 > 219
R: 252 > 226
R: 242 > 237
R: 203 > 201
R: 253 > 206
R: 211 > 210
```

```
R: 225 > 210
R: 251 > 216
R: 236 > 218
R: 236 > 218
R: 241 > 219
R: 252 > 226
R: 242 > 237
R: 203 > 201
R: 253 > 206
R: 211 > 210
R: 225 > 210
R: 251 > 216
R: 236 > 218
R: 236 > 218
R: 241 > 219
R: 252 > 226
R: 242 > 237
R: 203 > 201
R: 253 > 206
R: 211 > 210
R: 225 > 210
R: 251 > 216
R: 236 > 218
R: 236 > 218
R: 241 > 219
R: 252 > 226
R: 242 > 237
R: 203 > 201
R: 253 > 206
R: 211 > 210
R: 225 > 210
R: 251 > 216
R: 236 > 218
R: 236 > 218
R: 241 > 219
R: 252 > 226
R: 242 > 237
R: 203 > 201
R: 253 > 206
R: 211 > 210
R: 225 > 210
R: 251 > 216
R: 236 > 218
R: 236 > 218
R: 241 > 219
R: 252 > 226
R: 242 > 237
R: 203 > 201
```

```
R: 253 > 206
R: 211 > 210
R: 225 > 210
R: 251 > 216
R: 236 > 218
R: 236 > 218
R: 241 > 219
R: 252 > 226
R: 242 > 237
R: 203 > 201
R: 253 > 206
R: 211 > 210
R: 225 > 210
R: 251 > 216
R: 236 > 218
R: 236 > 218
R: 241 > 219
R: 252 > 226
R: 242 > 237
R: 203 > 201
R: 253 > 206
R: 211 > 210
R: 225 > 210
R: 251 > 216
R: 236 > 218
R: 236 > 218
R: 241 > 219
R: 252 > 226
R: 242 > 237
R: 203 > 201
R: 253 > 206
R: 211 > 210
R: 225 > 210
R: 251 > 216
R: 236 > 218
R: 236 > 218
R: 241 > 219
R: 252 > 226
R: 242 > 237
R: 203 > 201
R: 253 > 206
R: 211 > 210
R: 225 > 210
R: 251 > 216
R: 236 > 218
R: 236 > 218
R: 241 > 219
R: 252 > 226
```

```
R: 242 > 237
R: 203 > 201
R: 253 > 206
R: 211 > 210
R: 225 > 210
R: 251 > 216
R: 236 > 218
R: 236 > 218
R: 241 > 219
R: 252 > 226
R: 242 > 237
R: 203 > 201
R: 253 > 206
R: 211 > 210
R: 225 > 210
R: 251 > 216
R: 236 > 218
R: 236 > 218
R: 241 > 219
R: 252 > 226
R: 242 > 237
R: 203 > 201
R: 253 > 206
R: 211 > 210
R: 225 > 210
R: 251 > 216
R: 236 > 218
R: 236 > 218
R: 241 > 219
R: 252 > 226
R: 242 > 237
Assinatura Válida!
```

[ ]:

```

```

[ ]:

```

```