NewHope

May 22, 2020

1 Exercício 1 - NewHope

Neste exercício temos de implementar uma classe em Python que implemente um protótipo do esquema NewHope de acordo com a candidatura ao concurso NIST-PQC. Uma vez que os algoritmos e a forma como funcionam são públicos, o grupo apenas precisou de compreender como cada um deles funciona, transformando essa ideia para o SageMath.

[1]:
```
import hashlib
import random
import math
```

1.1 Parâmetros

[2]:
```
n = 1024
q = 12289
_7n_4=1792
_3n_8 = 384
q= 12289
k= 8



F = GF(q) ;  R = PolynomialRing(F, name="w")
w = (R).gen(); w = w

g = (w^n + 1)
xi = g.roots(multiplicities=False)[-1]
rs = [xi^(2*i+1)  for i in range(n)]
base = crt_basis([(w - r) for r in rs])
```

1.2 Funções auxiliares

[3]:
```
def hamming_weight( num):
    weight = 0

    while num:
        weight += 1
        num &= num - 1

    return weight

def bitRev( v):
    s = 0
    for i in range(0, int(math.log(n,2))):
        s += (((v >> i) & 1) << (math.log(n,2)-1-i))
    return s

def PolyBitRev( p):
    r  = [None] * n
    for i in range(0, n):
        r[bitRev(i)] = p[i]
    return r


def Poly_mult( c1, c2):
    r = [None] * n
    for i in range(0, n):
        r[i] =  (c1[i]*c2[i] % q)
    return r

def Poly_add( c1, c2):
    r = [None] * n
    for i in range(0, n):
        r[i] =  (c1[i]+c2[i] % q)
    return r

def PolySubtract(c1, c2):
    r = [None] * n
    for i in range(0, n):
        r[i] =  (c1[i]-c2[i] % q)
    return r
```

[4]:
```
'''
Number-theoretic transform
'''
def ntt(f):
    def _expand_(f):
```

```
        u = f.list()
        return u + [0]*(n-len(u))

    def _ntt_(xi,N,f):
        if N==1:
            return f
        N_ = N/2 ; xi2 =  xi^2
        f0 = [f[2*i]   for i in range(N_)] ; f1 = [f[2*i+1] for i in range(N_)]
        ff0 = _ntt_(xi2,N_,f0) ; ff1 = _ntt_(xi2,N_,f1)

        s  = xi ; ff = [F(0) for i in range(N)]
        for i in range(N_):
            a = ff0[i] ; b = s*ff1[i]
            ff[i] = a + b ; ff[i + N_] = a - b
            s = s * xi2
        return ff

    return _ntt_(xi,n,_expand_(f))

def ntt_inv(ff):
    return sum([ff[i]*base[i] for i in range(n)])
```

1.3 Algoritmos implementados de acordo com a documentação da candidatura

[5]:
```
'''
Algorithm 1 NewHope-CPA-PKE Key Generation
'''
def NewHope_CPA_PKE_Gen():
    seed = bytearray(os.urandom(32))
    z = hashlib.shake_256(b'0x01'+seed).digest(int(64))
    publicseed = z[0:32]
    noiseseed = z[32:64]
    _a =GenA(publicseed)
    s = PolyBitRev(Sample(noiseseed,0))
    e = PolyBitRev(Sample(noiseseed,1))
    _s = ntt(R(s))
    _e = ntt(R(e))
    aux =  Poly_mult(_a, _s)
    _b = Poly_add(aux, _e)
    print("Computing public key pk")
    pk = EncodePK( _b, publicseed)

    print("Computing secret key sk")
    sk =EncodePoly(_s)

    print("Public key generation complete. Returning keys")
```

```
    return pk, sk
```

[6]:
```
'''
Algorithm 2 NewHope-CPA-PKE Encryption
'''
def NewHope_CPA_PKE_Encrypt( pk, message, coin):
    _b, publicseed = DecodePK(pk)
    _a = GenA(publicseed)
    _s = PolyBitRev(Sample(coin,0))
    _e = PolyBitRev(Sample(coin,1))
    __e = Sample(coin,2)
    _t = ntt(R(_s))
    e_ntt = ntt(R(_e))
    __a = Poly_mult(_a, _t)
    u_hat = Poly_add( __a, e_ntt)

    v = EncodeMsg(message)

    bhat_that = Poly_mult(_b, _t)
    ntt_temp = ntt_inv(bhat_that)

    sum1 = Poly_add(ntt_temp, __e)
    v_prime = Poly_add(sum1, v)
    h = Compress(v_prime)
    c = EncodeC(u_hat, h)
    return c
```

[7]:
```
'''
Algorithm 3 NewHope-CPA-PKE Decryption
'''
def NewHope_CPA_PKE_Decrypt( c, sk):
    print("========================= Decrypting Message␣
 ↪=========================")
    u_hat, h = DecodeC(c)
    s_hat = DecodePoly(sk)
    v_prime = Decompress(h)
    us_product = Poly_mult(u_hat, s_hat)
    inv_product = ntt_inv(us_product)
    v_sub = PolySubtract(v_prime, inv_product)
    m = DecodeMsg(v_sub)
    return m
```

[8]:
```
'''
Algorithm 4 Deterministic sampling of polynomials in R q from   8 n
'''
def Sample( seed, nounce):
```

```
    r = [None] * n
    extseed = bytearray(random.sample(range(0, 256), 34))
    extseed[0:32] = seed[0:32]
    extseed[32] = nounce
    for i in range (0,int(n/64)):
        extseed[33] = i
        buf = hashlib.shake_256(extseed).digest(int(128))
        for j in range(0,64):
            a = buf[2*j]
            b = buf[2*j+1]
            val = hamming_weight(a) + q - (hamming_weight(b) % q)
            r[i*64+j] = val
    return r
```

[9]:
```
'''
Algorithm 5 Deterministic generation of â by expansion of a seed
'''
def GenA(seed):
    a = [None] * n
    extseed = bytearray(random.sample(range(0, 256), 33))
    extseed[0:32] = seed[0:32]
    for i in range(0, int(n/64)):
        ctr = 0
        extseed[32] = i
        state = hashlib.shake_128(extseed)
        while ctr < 64:
            buf = state.digest(int(168))
            j = 0
            while j<168 and ctr<64:
                val = buf[j] | (buf[j+1] << 8)
                if val < 5*q:
                    a[i*64+ctr] = val
                    ctr += 1
                j += 2
    return a
```

[10]:
```
'''
Algorithm 6 Encoding of a polynomial in R q to a byte array
'''
def EncodePoly(s):
    r = [None]*_7n_4
    for i in range(0, 256):
        t0 = int( s[(4*i)+0] % q )
        t1 = int( s[(4*i)+1] % q )
        t2 = int( s[(4*i)+2] % q )
        t3 = int( s[(4*i)+3] % q )
        r[(7*i)+0] = t0 & int(0xff)
```

```
        r[(7*i)+1] = (t0 >> 8) | ((t1 << 6)%4294967296) & int(0xff)
        r[(7*i)+2] = (t1 >> 2) & int(0xff)
        r[(7*i)+3] = (t1 >> 10) | ((t2 << 4)%4294967296) & int(0xff)
        r[(7*i)+4] = (t2 >> 4) & int(0xff)
        r[(7*i)+5] = (t2 >> 12) | ((t3 << 2)%4294967296) & int(0xff)
        r[(7*i)+6] = (t3 >> 6) & int(0xff)
    return r
```

[11]:

```
'''
Algorithm 7 Decoding of a polynomial represented as a byte array into an element␣
↪in R q
'''
def DecodePoly( v):
    print('Starting decoding polynomial')
    r = [None]*n
    for i in range(0, 256):
        r[(4*i)+0] = int(v[(7*i)+0]) |␣
↪(((int(v[(7*i)+1])&int(0x3f))<<8)%4294967296)
        r[(4*i)+1] = (int(v[(7*i)+1]) >> 6) | ((int(v[(7*i)+2]) <<␣
↪2)%4294967296) | (((int(v[(7*i)+3])&int(0x0f))<<10)%4294967296)
        r[(4*i)+2] = (int(v[(7*i)+3]) >> 4) | ((int(v[(7*i)+4]) <<␣
↪4)%4294967296) | (((int(v[(7*i)+5])&int(0x03))<<12)%4294967296)
        r[(4*i)+3] = (int(v[(7*i)+5]) >> 2) | ((int(v[(7*i)+6]) << 6)%4294967296)
    print('Done decoding polynomial')
    return r
```

[12]:

```
'''
Algorithm 8 Encoding of the public key
'''
def EncodePK( b_hat, publicseed):
    r = [None]*int(_7n_4 + 32)
    r[0:_7n_4] = EncodePoly(b_hat)
    r[_7n_4:] = publicseed
    return r
```

[13]:

```
'''
Algorithm 9 Decoding of the public key
'''
def DecodePK( pk):
    print('Starting decoding public key')
    b_hat = DecodePoly(pk[0:_7n_4])
    seed = pk[_7n_4:]
    print('Done decoding public key')
    return b_hat, seed
```

[14]:
```
'''
Algorithm 10 Message encoding
'''
def EncodeMsg( m):
    v = [None]*n
    for i in range(0, 32):
        for j in range(0, 8):
            mask = int(-((int(m[i]>>j))&int(1)))
            v[(8*i)+j+0] = int(mask&(q//2))
            v[(8*i)+j+256] = int(mask&(q//2))
            v[(8*i)+j+512] = int(mask&(q//2))
            v[(8*i)+j+768] = int(mask&(q//2))
    return v
```

[15]:
```
'''
Algorithm 11 Message decoding
'''
def DecodeMsg( v):
    m = [0]*32
    for i in range(0, 256):
        t = abs(int((v[i+0])%q) - int((q)//2))
        t = t + abs(int(((v[i+256])%q) - int((q)//2)))
        t = t + abs(int(((v[i+512])%q) - int((q)//2)))
        t = t + abs(int(((v[i+768])%q) - int((q)//2)))
        t = t - q
        t = t >> 15
        m[i>>3] = int(m[i>>3]) | -(int(t)<<int(i&7))
    return m
```

[16]:
```
'''
Algorithm 12 Ciphertext compression
'''
def Compress( v):
    k = 0
    t = [None]*8
    h = [None]*_3n_8
    for l in range(0, 128):
        i = 8*l
        for j in range(0, 8):
            t[j] = int(v[i+j] % q)
            t[j] = (((int(t[j]<<3))+q//2)//q) & int(0x7)
        h[k+0] = (t[0] | ((t[1]<<3)) | ((t[2]<<6))) #%256
        h[k+1] = ((t[2]>>2) | ((t[3]<<1)) | ((t[4]<<4)) | ((t[5]<<7))) #%256
        h[k+2] = ((t[5]>>1) | ((t[6]<<2)) | ((t[7]<<5))) #%256
        k += 3
    return h
```

[17]:
```
'''
Algorithm 13 Ciphertext encoding
'''
def EncodeC(u, h):
    c = [None]*(_7n_4 + _3n_8)
    c[0:_7n_4] = EncodePoly(u)
    c[_7n_4:] = h
    return c
```

[18]:
```
'''
Algorithm 14 Ciphertext decoding
'''
def DecodeC( c):
    u = DecodePoly(c[0:_7n_4])
    h = c[_7n_4:]
    return u, h
```

[19]:
```
'''
Algorithm 15 Ciphertext decompression
'''
def Decompress( h):
    r = [None]*n
    k = 0
    for l in range(0, 128):
        i = 8*l
        r[i+0] = h[k+0] & 7
        r[i+1] = (h[k+0]>>3) & 7
        r[i+2] = (h[k+0]>>6) | (((h[1]<<2))&4)
        r[i+3] = (h[k+1]>>1) & 7
        r[i+4] = (h[k+1]>>4) & 7
        r[i+5] = (h[k+1]>>7) | (((h[2]<<1))&6)
        r[i+6] = (h[k+2]>>2) & 7
        r[i+7] = (h[k+2]>>5)
        k = k +3
        for j in range(0, 8):
            r[i+j] = (((r[i+j])*q)+4)>>3
    return r
```

1.4 Testes

[20]:
```
pk, sk = NewHope_CPA_PKE_Gen()
```

```
Computing public key pk
Computing secret key sk
Public key generation complete. Returning keys
```

[21]:
```python
coin = bytearray(random.sample(range(0, 256), 32))
message = [225, 235, 49, 214, 170, 104, 167, 11, 44, 191, 245, 93, 225, 169,␣
↪110, 109, 210, 245, 50, 76, 61, 222, 120, 169, 152, 103, 251, 147, 188, 248,␣
↪161, 144]
c = NewHope_CPA_PKE_Encrypt(pk, message, coin)
t = NewHope_CPA_PKE_Decrypt(c, sk)
print("========================== Here is the original message and recovered␣
↪message ==========================")
print(message)
print(t)
```

```
Starting decoding public key
Starting decoding polynomial
Done decoding polynomial
Done decoding public key
========================== Decrypting Message ==========================
Starting decoding polynomial
Done decoding polynomial
Starting decoding polynomial
Done decoding polynomial
========================== Here is the original message and recovered message
==========================
[225, 235, 49, 214, 170, 104, 167, 11, 44, 191, 245, 93, 225, 169, 110, 109,
210, 245, 50, 76, 61, 222, 120, 169, 152, 103, 251, 147, 188, 248, 161, 144]
[64, 32, 0, 16, 2, 8, 4, 0, 0, 49, 0, 64, 97, 32, 40, 13, 0, 0, 0, 64, 32, 132,
64, 160, 8, 4, 168, 0, 0, 0, 0, 0]
```

[ ]:
```python
m = [225, 235, 49, 214, 170, 104, 167, 11, 44, 191, 245, 93, 225, 169, 110, 109,␣
↪210, 245, 50, 76, 61, 222, 120, 169, 152, 103, 251, 147, 188, 248, 161, 144]
v_prime = EncodeMsg(m)
m_= DecodeMsg(v_prime)
if(m_ == m):
        print("trueeeee")
for i in range(0,1):
    c = Compress(v_prime)
    print(c)
    print(v_prime)
    m__ = Decompress(c)
    print(m__)
    if(v_prime == m__):
        print("trueeeee")
    f = DecodeMsg(m__)
    if (f== m):
        print("niceee")
```

1.5 KEM-IND-CPA

[22]:

```python
'''
Algorithm 16 NewHope-CPA-KEM Key Generation
'''
def NewHope_CPA_KEM_Gen():
    pk, sk = NewHope_CPA_PKE_Gen()
    return (pk,sk)

'''
Algorithm 17 NewHope-CPA-KEM Encapsulation
'''
def NewHope_CPA_KEM_Encapsulation( pk):
    coin = (random.sample(range(0, 256), 32))
    z = hashlib.shake_256(b'0x02'+coin).digest(64)
    k = z[0:32]
    coin_hat = z[32:]
    c = NewHope_CPA_PKE_Encrypt(pk, k, coin_hat)
    ss = hashlib.shake_256(k).digest(32)

    return (c,ss)

'''
Algorithm 18 NewHope-CPA-KEM Decapsulation
'''
def NewHope_CPA_KEM_Decapsulation ( c,sk):
    k_hat= NewHope_CPA_PKE_Decrypt(c,sk)
    ss2 = hashlib.shake_256(k_hat).digest(32)
    return ss2
```