NTRU

Imports

In [1]:
```
from hashlib import sha256
```

NTRU

conv

Operação de multiplicação utilizada em NTRU. É identica à multiplicação polinomial, porém reduz o output "modulo x^n-1", ou seja, x^n é substituído por 1, x^(n+1) por x, x^(n+2) por x^2, etc. Os inputs são dois polinomios de n coeficientes, ou seja, com membros de 1 até x^(n-1). O output também é um polinomial com n coeficientes, pois os elementos x^n, x^(n+1), x^(n+2), etc. foram eliminados.

balancedmod

Esta função recebe como inputs um polinomial com n coeficientes e um integer positivo. O output é o mesmo polinomial, exceto que cada coeficiente é reduzido com módulo q.

randompoly

A função randompoly() retorna um polinomial, onde exatamente d coeficientes são diferentes de 0 (d não são 0, e n - d são 0). É de ter em conta que todos os coeficientes do polinomio estão limitados a 1 e -1.

invertmodprime

A função invertmodprime(f,p) recebe dois inputs: um polinomial com n coeficientes; um número primo p. O output é um polinómio g com n coeficientes de modo a que conv(f,g) seja 1 + p * u para um polinomial u. Esta função levanta uma exceção caso este polinómio g não exista.

invertmodpowerof2

Função semelhante à invertmodprime(f,p) com a diferença que o segundo argumento têm de ser múltiplo de 2.

keypair

Esta função retorna a chave pública NTRU h e as respectivas chaves secretas f,f3.

encrypt

A função de cifragem recebe uma mensagem e a chave pública e retorna a mensagem cifrada. O texto cifrado é h * r + m modulo q, onde m é a mensagem e r é um polinomio random.

decrypt

Função em que é decifrado o criptograma gerado pela cifragem de uma mensagem, recorrendo a uma chave privada.

wrap

Onde se processa o encapsulamento de uma chave, assim como a geração de um criptograma a partir da cifragem de um polinómio aleatório com uma chave pública. É aplicado o algoritmo *sha256* sobre o polinómio gerado.

unwrap

Onde se processa o desencapsulamento de uma chave, partindo de um criptograma gerado no encapsulamento e de uma chave privada.

Class NTRU

In [7]:
```
class NTRU:

    def __init__(self, n, d, q):
        self.n = n
        self.d = d
        self.q = q
        self.Zx = PolynomialRing(ZZ,'x')
        self.x = self.Zx.gen()




    def conv(self, f, g):
        res = (f * g) % (self.x^self.n-1)
        return res


    def balancedmod(self, poly, q):
        g = list(((poly[i] + q//2) % q) - q//2 for i in range(self.n))
        return self.Zx(g)


    def randompoly(self):
        assert self.d <= self.n
        result = self.n*[0]
        for j in range(self.d):
            while True:
                r = randrange(self.n)
                if not result[r]: break
            result[r] = 1-2*randrange(2)
        return self.Zx(result)


    def invertmodprime(self, f,p):
        T = self.Zx.change_ring(Integers(p)).quotient(self.x^self.n-1)
        return self.Zx(lift(1 / T(f)))


    def invertmodpowerof2(self, f):
        assert self.q.is_power_of(2)
        g = self.invertmodprime(f,2)
        while True:
            r = self.balancedmod(self.conv(g,f), self.q)
            if r == 1:
                return g
            g = self.balancedmod(self.conv(g,2 - r), self.q)


    def keypair(self):
        while True:
            try:
                f = self.randompoly()
                f3 = self.invertmodprime(f,3)
                fq = self.invertmodpowerof2(f)
                break
            except:
                pass
        g = self.randompoly()
        publickey = self.balancedmod(3 * self.conv(fq,g), self.q)
        secretkey = (f,f3)
        return (publickey,secretkey)


    def poly_to_string(self, poly):
        res = ""
        for el in poly:
            res = str(el) + res
        return res


    def messagepoly(self):
        result = list(randrange(3) - 1 for j in range(self.n))
        return self.Zx(result)


    def encrypt(self, message, publickey):
        r = self.randompoly()
        return self.balancedmod(self.conv(publickey,r) + message, self.q)


    def decrypt(self, ciphertext, secretkey):
        f, f3 = secretkey
        a = self.balancedmod(self.conv(ciphertext,f),self.q)
        return self.balancedmod(self.conv(a,f3),3)


    def wrap(self, publickey):
        r = self.randompoly()
        c = self.encrypt(r,publickey)
        k = sha256(self.poly_to_string(r).encode('utf-8')).hexdigest()
        return(c,k)


    def unwrap(self, secretkey, c):
        r = self.decrypt(c, secretkey)
        k1 = sha256(self.poly_to_string(r).encode('utf-8')).hexdigest()
        return k1
```

Testing PKE

In [10]:
```
N = NTRU(7, 5, 256)
(p,s) = N.keypair()
mess = N.messagepoly()
e = N.encrypt(mess, p)
l = N.decrypt(e, s)
print(mess == l)
```

```
True
```

Testing KEM

In [11]:
```
N = NTRU(7, 5, 256)
(p,s) = N.keypair()
wr, cc = N.wrap(p)
uwr = N.unwrap(s, wr)
print(uwr == cc)
```

```
True
```

Bibliografia:

https://ntru.org/

In [ ]:
```

```