# ntru

May 10, 2021

## 1 NTRU

[1]:
```
from random import choice
from cryptography.hazmat.primitives import hashes

N = 509
p = 3
#log_q = 11
#q = 1 << log_q # 2048
q = next_prime(2000)

Z.<w>  = ZZ[]
R.<w>   = QuotientRing(Z , Z.ideal(w^N - 1))

Q.<w>  = Integers(q)[]
Rq.<w>  = QuotientRing(Q , Q.ideal(w^N - 1))

# Mensagem é dada num intervalo de coeficientes [0, q-1],
# por isso o modulo dos valores tem de ser recentrados
# para o intervalo [-q/2, q/2-1].
def centered(l,p):
    fp = [ lift(Mod(a,p)) for a in l ]
    return [ u if (u <= p//2) else u-p for u in fp ]
```

`__init__(self, N, p, q)`

O construtor da classe NTRU vai instanciar os parametros e gerar as chaves publica e privada (`self.h` e `self.f` respetivamente).

`random_poly_ternary(self)`

Este metodo gera um polinómio ternário aleatorio

> **Nota:** Nesta implementação os dados (*plaintexts*, *ciphertexts*) estão sob a forma de polinómios, não são fornecidos funções auxiliares de *encoding* e *decoding*

[2]:
```
class NTRU:
    def __init__(self, N, p, q):
        self.N = N
        self.p = p
```

```
        self.q = q
        # KeyGen quando instancia NTRU
        # f tem de ser invertivel senão
        # o invert_of_unit podia lançar uma
        # excepção de divisão por 0
        while True:
            self.f = 1 + self.p * R(self.random_poly_ternary())
            if Rq(list(self.f)).is_unit():
                break
        self.g = self.p * R(self.random_poly_ternary())
        fq = Rq(list(self.f)).inverse_of_unit()
        hq = fq * Rq(list(self.g))
        self.h = R([lift(a) for a in list(hq)])

    def encrypt(self, msg):
        r = R(self.random_poly_ternary())
        m = R(msg)
        return centered(list(self.h*r + m), self.q)

    def decrypt(self, enc):
        e = R(enc)
        a = centered(list(self.f * e), self.q)
        return centered(list(R(a)),self.p)

    def encapsulate(self):
        rm = self.random_poly_ternary()
        sha3 = hashes.Hash(hashes.SHAKE256(int(256)))
        # + p // 2 porque nao dá para converter de imediato
        # valores inteiros negativos para complemento para 2
        # só sao considerados os valores unsigned
        sha3.update(bytes([ c + self.p // 2 for c in rm ]))
        shared_key = sha3.finalize()
        return (shared_key, self.encrypt(rm))

    def decapsulate(self,cipher):
        rm = self.decrypt(cipher)
        sha3 = hashes.Hash(hashes.SHAKE256(int(256)))
        # + p // 2 porque nao dá para converter de imediato
        # valores inteiros negativos para complemento para 2
        # só sao considerados os valores unsigned
        sha3.update(bytes([ c + self.p // 2 for c in rm ]))
        return sha3.finalize()

    def random_poly_ternary(self):
        return [ choice([-1,0,1]) for i in range(self.N) ]

ntru = NTRU(N,p,q)
```

```python
# NTRU PKE Test
msg = ntru.random_poly_ternary()
enc = ntru.encrypt(msg)
dec = ntru.decrypt(enc)
print("NTRU PKE Test:",msg == dec)

# NTRU KEM Test
bob_shared_secret, enc = ntru.encapsulate()
alice_shared_secret = ntru.decapsulate(enc)
print("NTRU KEM Test:",bob_shared_secret == alice_shared_secret)
```

```
NTRU PKE Test: True
NTRU KEM Test: True
```