Trabalho Prático 2 - BIKE - Grupo 18

Introdução ao problema

Neste problema foi-nos indicado para:

- Criar um protótipo em Sagemath da técnica de criptografia pós-quântica BIKE, implementando um KEM IND-CPA seguro e um PKE IND-CCA seguro.

Resolução do problema

Imports

In [ ]:
```
import random, hashlib
```

In [ ]:
```
class KEM_BIKE:

    def __init__(self, N, R, W, T):

        self.r = R
        self.n = N
        self.w = W
        self.t = T

        self.k2 = GF(2)

        F.<x> = PolynomialRing(self.k2)
        R.<x> = QuotientRing(F, F.ideal(x^self.r + 1))
        self.R = R

    def polynom_to_vector_size_r(self, pol):

        v = VectorSpace(self.k2, self.r)

        return v(pol.list() + [0]*(self.r - len(pol.list())))

    def polynom_tuple_to_vector_size_n(self, pol_t):

        v = VectorSpace(self.k2, self.n)

        return v( self.polynom_to_vector_size_r(pol_t[0]).list() +
                  self.polynom_to_vector_size_r(pol_t[1]).list() )

    def hamming_weight(self, vec):

        sum = 0

        for x in vec:
            if x == self.k2(1):
                sum += 1

        return sum
```

```
    def hash_e(self, e0, e1):

        hash = hashlib.sha3_256()
        hash.update(e0.encode())
        hash.update(e1.encode())

        return hash.digest()

    def rot_matrix(self, vec):

        M = Matrix(self.k2, self.r, self.r)
        M[0] = self.polynom_to_vector_size_r(vec)

        for i in range(1, self.r):

            v = VectorSpace(self.k2, self.r)
            vec_aux = v()
            vec_aux[0] = M[i-1][-1]

            for j in range(self.r-1):

                vec_aux[j+1] = M[i-1][j]

            M[i] = vec_aux

        return M

    def bit_flip(self, h, y, s):

        iter_nr = self.r

        x = y
        z = s

        while self.hamming_weight(z) > 0 and iter_nr > 0:
            print('iter', iter_nr)

            ham_weights = [ self.hamming_weight( z.pairwise_product(h[i]) ) for i in
            max_weight = max(ham_weights)

            for i in range(self.n):
                if ham_weights[i] == max_weight:
                    x[i] += self.k2(1)
                    z += h[i]

            iter_nr = iter_nr - 1

        if iter_nr == 0:
            raise ValueError("The limit of iterations was reached!")

        return x

    def coeficient_generator(self, w, n):

        val = [1] * w + [0] * (n-w-2)
        random.shuffle(val)

        return self.R([1]+val+[1])

    def generate_key_pair(self):

        h0 = self.coeficient_generator(self.w // 2, self.r)
        h1 = self.coeficient_generator(self.w // 2, self.r)
```

```
        g = self.coeficient_generator(self.r // 2, self.r)

        f0 = g * h1
        f1 = g * h0

        return { 'pub_key' : (h0, h1),
                 'priv_key' : (f0, f1)}

    def encapsulate(self, pub_key):

        val = [1]*self.t + [0]*(self.n - self.t)
        random.shuffle(val)
        (e0, e1) = (self.R(val[:self.r]), self.R(val[self.r:]))

        m = self.R.random_element()

        k = self.hash_e(str(e0), str(e1))
        c = (m * pub_key[0] + e0, m * pub_key[1] + e1)

        return (k, c)

    def decapsulate(self, priv_key, c):

        code = self.polynom_tuple_to_vector_size_n(c)
        h = block_matrix(2, 1, [self.rot_matrix(priv_key[0]), self.rot_matrix(priv_k
        s = code * h

        s_bf = self.bit_flip(h, code, s)

        (s0, s1) = ( self.R(s_bf.list()[:self.r]), self.R(s_bf.list()[self.r:]) )

        e = (c[0] - s0 * 1, c[1] - s0 * priv_key[0]/priv_key[1])


        e_weight = self.hamming_weight(self.polynom_to_vector_size_r(e[0])) + self.h
        if e_weight != self.t:
            raise ValueError("Decoding error!")

        k = self.hash_e(str(e[0]), str(e[1]))

        return k
```

In [ ]:

```
R = 11
N = 2 * R
W = 142
T = 134

kem_bike = KEM_BIKE(N, R, W, T)

key_pair = kem_bike.generate_key_pair()

(k,c) = kem_bike.encapsulate(key_pair['pub_key'])

k_l = kem_bike.decapsulate(key_pair['priv_key'], c)

if k == k_l:
    print("The decapsulated key is equal to the original one!")
else:
    print("The decapsulated key doesn't match with the original one!")
```