TP3 - BIKE

May 3, 2023

1 Trabalho Prático 3

Trabalho realizado pelo grupo 11:

- Beatriz Fernandes Oliveira, PG50942
- Bruno Filipe Machado Jardim, PG49997

1.1 Algoritmo BIKE

Nesta segunda parte do trabalho prático 3, foi-nos pedido:

1. A criação de um protótipo em Sagemath para o algoritmo BIKE, que se baseia no problema da descodificação de códigos lineares de baixa densidade que são simples de implementar.
2. E, posteriormente, para essa técnica pretende-se implementar um KEM, que seja IND-CPA seguro, e um PKE que seja IND-CCA seguro.

[15]:
```
import random
from cryptography.hazmat.primitives import hashes
import os
from cryptography.hazmat.primitives.ciphers import Cipher, algorithms, modes
```

[82]:
```
class BIKE_KEM(object):

    def __init__(self, r, N, T, L):
        self.r = r
        self.n = N
        self.t = T
        self.l = L
        self.q = GF(2) # corpo finito de tamanho 2
        F.<x> = PolynomialRing(self.q)
        R.<x> = QuotientRing(F, F.ideal(x^self.r + 1))
        self.R = R

    def hashed(self, e0, e1):
        digest = hashes.Hash(hashes.SHA256())
        digest.update(e0.encode())
        digest.update(e1.encode())
        return digest.finalize()
```

```
    def Hamming(self, x):
        return sum([1 if a == self.q(1) else 0 for a in x])

    def ppairwise(self, x1, x2):
        return x1.pairwise_product(x2)

    def geraCoefs(self, sparse=3):
        coefs = [1]*sparse + [0]*(self.r - 2 - sparse)
        random.shuffle(coefs)
        return self.R([1] + coefs + [1])

    def geraError(self):
        err = [1]*self.t + [0]*(self.n - self.t)
        random.shuffle(err)
        return self.R(err[:self.r]),self.R(err[self.r:])

    def keyGenerator(self):
        h0 = self.geraCoefs()
        h1 = self.geraCoefs()
        return (h0, h1), (1, h0/h1)

    def encaps(self, public):
        e0, e1 = self.geraError()
        key = self.hashed(str(e0), str(e1))
        r = self.R.random_element()
        c = (r * public[0] + e0, r * public[1] + e1)
        return key, c

    def to_Vector_r(self, f):
        fl = f.list()
        V = VectorSpace(self.q, self.r)
        return V(fl + [0]*(self.r - len(fl)))

    def to_Vector_n(self, c):
        f = self.to_Vector_r(c[0]).list() + self.to_Vector_r(c[1]).list()

        V = VectorSpace(self.q, self.n)
        return V(f)

    def rotate_unit(self, unit):
        V = VectorSpace(self.q, self.r)
        v = V()
        v[0] = unit[-1]
        for i in range(self.r-1):
            v[i+1] = unit[i]
        return v
```

```
    def rotate(self, h):
        matrix = Matrix(self.q, self.r, self.r)
        matrix[0] = self.to_Vector_r(h)

        for i in range(1, self.r):
            matrix[i] = self.rotate_unit(matrix[i-1])

        return matrix

    def bitFlip(self, matrix, c, s):
        c_ = c
        _s = s
        nItr = self.r

        while self.Hamming(_s) > 0 and nItr > 0:
            nItr -= 1
            pesos = [self.Hamming(self.ppairwise(_s, matrix[i])) for i in␣
↪range(self.n)]
            maximo = max(pesos)

            for j in range(self.n):
                if pesos[j] == maximo:
                    c_[j] = 1 - c_[j]
                    _s += matrix[j]

        if nItr == 0:
            return None

        return c_

    def decaps(self, private, c):

        private_rotate = (self.rotate(private[0]), self.rotate(private[1]))
        matrix = block_matrix(2,1, [private_rotate[0], private_rotate[1]])

        c_vector = self.to_Vector_n(c)
        syndrome = c_vector * matrix
        error = self.bitFlip(matrix, c_vector, syndrome)

        if error == None:
            print("Máximo de iterações atingidas")
            return None
        else:
            elist = error.list()
            error0, error1 = self.R(elist[:self.r]), self.R(elist[self.r:])
            e0, e1 = c[0] - error0, c[1] - error1
```

```
            if self.Hamming(self.to_Vector_r(e0)) + self.Hamming(self.
↪to_Vector_r(e1)) != self.t:
                print("Erro")
                return None

        return self.hashed(str(e0), str(e1))
```

1.1.1 Exemplo

[67]:
```
bike = BIKE_KEM(257, 514, 16, 256) #r, n, t, l

private, public = bike.keyGenerator()

key_encaps, c = bike.encaps(public)

key_decaps = bike.decaps(private, c)

if key_encaps == key_decaps and key_decaps != None:
    print("A chave desencapsulada é igual à chave encapsulado. O algoritmo BIKE␣
↪como um KEM funciona!")
```

```
A chave desencapsulada é igual à chave encapsulado. O algoritmo BIKE como um KEM
funciona!
```

1.2 Implementação do PKE

- **Cifragem**: $E(x) \equiv \vartheta r \leftarrow h \,.\, \vartheta y \leftarrow (x \oplus g(r) \,.\, (e,k) \leftarrow f(y||r) \,.\, \vartheta c \leftarrow k \oplus r \,.\, (y,e,c)$
- **Desencapsulamento**: $D(y,e,c) \equiv \vartheta k \leftarrow KREv(e) \,.\, \vartheta r \leftarrow c \oplus k \,.\, if(e,k) \neq f(y||r)\ then\ \perp\ else\ y \oplus g(r)$

[1]:
```
class BIKE_PKE(object):

    def __init__(self, R, N, T, L):
        self.kem = BIKE_KEM(R, N, T, L)
        self.r = self.kem.r
        self.n = self.kem.n
        self.t = self.kem.t
        self.l = self.kem.l
        self.q = self.kem.q
        self.R = self.kem.R

    def hashed(self, r):
        digest = hashes.Hash(hashes.SHA256())
        digest.update(str(r).encode())
        return digest.finalize()
```

```
    def keyGenerator(self):
        self.private, self.public = self.kem.keyGenerator()
        return self.private, self.public

    def f(self, public, m, e0, e1):
        w = (m * public[0] + e0, m * public[1] + e1)
        key = self.kem.hashed(str(e0), str(e1))
        return (key, w)

    def xor(self, x, y):
        bits = b''
        x_len = len(x)
        y_len = len(y)
        i=0
        while i < x_len:
            for j in range(y_len):
                if i < x_len:
                    bits += (x[i]^^y[j]).to_bytes(1, byteorder='big')
                    i += 1
                else:
                    break
        return bits

    def desencaps_key(self, e0, e1):
        if self.kem.Hamming(self.kem.to_Vector_r(e0)) + self.kem.Hamming(self.
↪kem.to_Vector_r(e1)) != self.t:
            print("Error")
            return None

        return self.kem.hashed(str(e0), str(e1))

    def decapsError(self,private, e):

        private_rotate = self.kem.rotate(private[0]), self.kem.
↪rotate(private[1])
        matrix = block_matrix(2,1,[private_rotate[0], private_rotate[1]])

        e_vector = self.kem.to_Vector_n(e)
        s = e_vector * matrix
        error = self.kem.bitFlip(matrix, e_vector, s)

        if error == None:
            print("Iterações atingiram o limite")
            return None
        else:
            listError = error.list()
```

```
            error0, error1 = self.R(listError[:self.r]), self.R(listError[self.
↪r:])
            e0, e1 = e[0] - error0, e[1] - error1

        return e0,e1

    def encrypt(self, msg, public):
        e0, e1 = self.kem.geraError()
        r = self.R.random_element()
        g = self.hashed(r)

        y = self.xor(msg.encode(), g)
        y_binary = bin(int.from_bytes(y, byteorder = sys.byteorder))
        R_y_binary = self.R(y_binary)

        (key, e) = self.f(public, R_y_binary + r, e0, e1)
        c = self.xor(str(r).encode(), key)

        return y, e, c

    def decrypt(self, private, y, e, c):

        e0, e1 = self.desencaps_error(private,e)
        k = self.desencaps_key(e0,e1)

        r_xored = self.xor(c,k)
        r = self.R(r_xored.decode())

        y_binary = bin(int.from_bytes(y, byteorder=sys.byteorder))
        R_y_Binary = self.R(y_binary)

        if (k,e) != self.f(self.public, R_y_Binary + r, e0, e1):
            print("Erro")
            return None
        else:
            hashed = self.hashed(r)
            plaintext = self.xor(y, hashed)

        return plaintext
```

1.2.1 Exemplo

[81]:
```
bike = BIKE_PKE(257, 514, 16, 256) #r, n, t, l

mensagem = "EC-TP3 Implementação do algoritmo BIKE com um PKE"

private, public = bike.keyGenerator()
```

```
msg_encp, key_encp, ciphertext = bike.encrypt(mensagem, public)

plaintext = bike.decrypt(private, msg_encp, key_encp, ciphertext)
plaintext.decode()
```

[81]: 'EC-TP3 Implementação do algoritmo BIKE com um PKE'