Trabalho Prático 2

Grupo 17, constituído por:

-- Joana Castro e Sousa, PG47282

-- Tiago Taveira Gomes, PG47702

-- João Carlos Pereira Rodrigues, PG46534

KYBER

> Kyber (formerly known as New Hope) is among the first post-quantum schemes to be standardized and already found its way into products. As a lattice-based system, Kyber is fast and its security guarantees are linked to an NP-hard problem. Also, it has all the nice mathematical ingredients to confuse the hell out of you: vectors of odd-looking polynomials, algebraic rings, error terms and a security reduction to "module lattices".
>
> https://media.ccc.de/v/rc3-2021-cwtv-230-kyber-and-post-quantum

---

Deste modo, iremos apresentar duas implementações (com recurso ao *SageMath* deste algoritmo: KYBER-CPAPKE e KYBER-KEM. Estas versões são apreentadas no documento oficial da 3a ronda (https://pq-crystals.org/kyber/data/kyber-specification-round3.pdf).

> Kyber is an IND-CCA2-secure key-encapsulation mechanism (KEM), which has first been described in [24]. The security of Kyber is based on the hardness of solving the learning-with-errors problem in module lattices (MLWE problem [66]). The construction of Kyber follows a two-stage approach: we first introduce an INDCPA-secure public-key encryption scheme encrypting messages of a fixed length of 32 bytes, which we call Kyber.CPAPKE. We then use a slightly tweaked Fujisaki–Okamoto (FO) transform [46] to construct the IND-CCA2-secure KEM. Whenever we want to emphasize that we are speaking about the IND-CCA2-secure KEM, we will refer to it as Kyber.CCAKEM.

KYBER-CPAPKE e KYBER-CCAKEM

KYBER-CPAPKE

Esta versão permite obter uma segurança do tipo IND-CPA (segurança contra ataques Chosen Plaintext Attacks).

KYBER-CCAKEM

Esta versão permite obter uma segurança do tipo IND-CCA (segurança contra ataques Chosen Ciphertext Attacks).

Numa primeira fase foi necessário implementar algumas funções auxiliares.

In [ ]:
```
# Créditos da implementação: https://github.com/fvirdia/lwe-on-rsa-copro
import sys
from sage.all import parent, ZZ, vector, PolynomialRing, GF
from sage.all import randint, set_random_seed, random_vector, matrix

# Funçåo auxiliar para determinar um valor de uma distribuição polinomial, dado um limite
def BinomialDistribution(eta):
    r = 0
    for i in range(eta):
        r += randint(0, 1) - randint(0, 1)
    return r


# Calcular a representação de `e`, com elementos entre `-q/2` and `q/2`
def balance(e, q=None):
    # e: a vector, polynomial or scalar
    # q: optional modulus, if not present this function tries to recover it from `e`
    # returns: a vector, polynomial or scalar over/in the integers
    try:
        p = parent(e).change_ring(ZZ)
        return p([balance(e_, q=q) for e_ in e])
    except (TypeError, AttributeError):
        if q is None:
            try:
                q = parent(e).order()
            except AttributeError:
                q = parent(e).base_ring().order()
        e = ZZ(e)
        e = e % q
        return ZZ(e-q) if e>q//2 else ZZ(e)
```

De seguida, a implementação da classe Kyber, que permite fornecer todos os métodos para ambas as versões implementadas, com a referida documentação:

In [ ]:
```
class Kyber:

    n = 256
    q = 7681
    eta = 4
    k = 3
    D = staticmethod(BinomialDistribution)
    f = [1]+[0]*(n-1)+[1]
    ce = n

    @classmethod
    # Gerar um par de chaves (pública e privada)
    def key_gen(cls, seed=None):
        # param cls: Kyber class, inherit and change constants to change defaults
        # param seed: seed used for random sampling if provided

        # Algoritmo baseado do Algoritmo 1 do documento especificado do Kyber

        n, q, eta, k, D = cls.n, cls.q, cls.eta, cls.k, cls.D

        if seed is not None:
            set_random_seed(seed)
```

```
        R, x = PolynomialRing(ZZ, "x").objgen()
        Rq = PolynomialRing(GF(q), "x")
        f = R(cls.f)

        A = matrix(Rq, k, k, [Rq.random_element(degree=n-1) for _ in range(k*k)])
        s = vector(R, k, [R([(D(eta)) for _ in range(n)]) for _ in range(k)])
        e = vector(R, k, [R([(D(eta)) for _ in range(n)]) for _ in range(k)])
        t = (A*s + e)  % f  # NOTE ignoring compression

        return (A, t), s

    @classmethod
    # IND-CPA cifragem sem compressão de dados
    def enc(cls, pk, m=None, seed=None):

        # param cls: Kyber class, inherit and change constants to change defaults
        # param pk: public key
        # param m: optional message, otherwise all zero string is encrypted
        # param seed: seed used for random sampling if provided

        # Algoritmo baseado do Algoritmo 2 do documento especificado do Kyber

        n, q, eta, k, D = cls.n, cls.q, cls.eta, cls.k, cls.D

        if seed is not None:
            set_random_seed(seed)

        A, t = pk

        R, x = PolynomialRing(ZZ, "x").objgen()
        f = R(cls.f)

        r  = vector(R, k, [R([(D(eta)) for _ in range(n)]) for _ in range(k)])
        e1 = vector(R, k, [R([(D(eta)) for _ in range(n)]) for _ in range(k)])
        e2 = R([(D(eta)) for _ in range(n)])

        if m is None:
            m = (0,)

        u = (r*A + e1) % f  # NOTE ignoring compression
        u.set_immutable()
        v = (r*t + e2 + q//2 * R(list(m))) % f  # NOTE ignoring compression
        return u, v

    @classmethod
    # IND-CPA decifragem
    def dec(cls, sk, c, decode=True):

        # param cls: Kyber class, inherit and change constants to change defaults
        # param sk: secret key
        # param c: ciphertext
        # param decode: perform final decoding

        # Algoritmo baseado do Algoritmo 3 do documento especificado do Kyber

        n, q = cls.n, cls.q

        s = sk
```

```
        u, v = c

        R, x = PolynomialRing(ZZ, "x").objgen()
        f = R(cls.f)

        m = (v - s*u) % f
        m = list(m)
        while len(m) < n:
            m.append(0)

        m = balance(vector(m), q)

        if decode:
            return cls.decode(m, q, n)
        else:
            return m

    @staticmethod
    # Decode vector `m` to `{0,1}^n` depending on distance to `q/2`
    def decode(m, q, n):

        # param m: a vector of length `leq n`
        # param q: modulus

        return vector(GF(2), n, [abs(e)>q/ZZ(4) for e in m] + [0 for _ in range(n-len(m))])

    @classmethod
    # IND-CCA encapsulamento sem compressão nem hash extra
    def encap(cls, pk, seed=None):

        # param cls: Kyber class, inherit and change constants to change defaults
        # param pk: public key
        # param seed: seed used for random sampling if provided

        # Algoritmo baseado do Algoritmo 4 do documento especificado do Kyber

        n = cls.n

        if seed is not None:
            set_random_seed(seed)

        m = random_vector(GF(2), n)
        m.set_immutable()
        set_random_seed(hash(m))  # NOTE: this is obviously not faithful

        K_ = random_vector(GF(2), n)
        K_.set_immutable()
        r = ZZ.random_element(0, 2**n-1)

        c = cls.enc(pk, m, r)

        K = hash((K_, c))  # NOTE: this obviously isn't a cryptographic hash
        return c, K

    @classmethod
    # IND-CCA desencapsulamento
    def decap(cls, sk, pk, c):

        # param cls: Kyber class, inherit and change constants to change defaults
```

```
        # param sk: secret key
        # param pk: public key
        # param c: ciphertext

        # Algoritmo baseado do Algoritmo 5 do documento especificado do Kyber

        n = cls.n

        m = cls.dec(sk, c)
        m.set_immutable()
        set_random_seed(hash(m))  # NOTE: this is obviously not faithful

        K_ = random_vector(GF(2), n)
        K_.set_immutable()
        r = ZZ.random_element(0, 2**n-1)

        c_ = cls.enc(pk, m, r)

        if c == c_:
            return hash((K_, c))  # NOTE: this obviously isn't a cryptographic hash
        else:
            return hash(c)  # NOTE ignoring z
```

Funções para facilitar uma série de testes

In [ ]:

```
# Testar a implementação de IND-CPA
def test_kyber_cpa(cls=Kyber, t=16):
    """
    Test correctness of IND-CPA encryption/decryption.
    TESTS::
        sage: test_kyber_cpa(Kyber)
    .. note :: An ``AssertionError`` if decrypted plaintext does not match original.
    """
    for i in range(t):
        # gerar chaves
        pk, sk = cls.key_gen(seed=i)
        # gerar uma mensagem aleatória (random_vector)
        m0 = random_vector(GF(2), cls.n)
        # print("mensagem: ", m0)
        # cifragem
        c = cls.enc(pk, m0, seed=i)
        # decifragem
        m1 = cls.dec(sk, c)
        # asserção
        assert(m0 == m1)


# Testar a implementação de IND-CCA
def test_kyber_cca(cls=Kyber, t=16):
    """
    Test correctness of IND-CCA encapsulation/decapsulation.
    TESTS::
        sage: test_kyber_cca(Kyber)
    .. note :: An ``AssertionError`` if final key does not match original.
    """
    for i in range(t):
        # gerar chaves
```

```
        pk, sk = cls.key_gen(seed=i)
        # encapsulamento
        c, K0 = cls.encap(pk, seed=i)
        # desencapsulamento
        K1 = cls.decap(sk, pk, c)
        # asserção
        assert(K0 == K1)


# Testar ambas as implementações
def test_kyber(cls=Kyber, t=16):
    """
    Test correctness of Kyber implementation.
    TESTS::
        sage: test_kyber(Kyber)
        <Kyber> CPA pass
        <Kyber> CCA pass
    """
    # testar IND-CPA
    print("<%s> IND-CPA"%(cls.__name__), end=" ")
    sys.stdout.flush()
    test_kyber_cpa(cls, t)
    # funcionou? Pass
    print("pass")

    # testar IND-CCA
    print("<%s> IND-CCA"%(cls.__name__), end=" ")
    sys.stdout.flush()
    test_kyber_cca(cls, t)
    # funcionou? Pass
    print("pass")
```

In [ ]:

```
print("Testar com apenas um teste:")
test_kyber(Kyber, 1)

print("Testar com 10 testes:")
test_kyber(Kyber, 10)

print("Testar com 20 testes:")
test_kyber(Kyber, 20)

print("Testar com 50 testes:")
test_kyber(Kyber, 50)
```

```
Testar com apenas um teste:
<Kyber> IND-CPA pass
<Kyber> IND-CCA pass
Testar com 10 testes:
<Kyber> IND-CPA pass
<Kyber> IND-CCA pass
Testar com 20 testes:
<Kyber> IND-CPA pass
<Kyber> IND-CCA pass
Testar com 50 testes:
<Kyber> IND-CPA pass
<Kyber> IND-CCA pass
```