Exercício 1 (Dilithium) - Trabalho Prático 3

Grupo 6:

Ruben Silva - pg57900

Luís Costa - pg55970

Resolução do Dilithium

Repositório Educativo

Material de Estruturas Criptográficas - Universidade Minho

imports

In [218…

```
import os
import random
from xoflib import shake128, shake256
from Crypto.Cipher import AES
from typing import Optional
```

Funções Auxiliares

In [219…

```
def reduce_mod_pm(x, n):
    x = x % n
    if x > (n >> 1):
        x -= n
    return x


def decompose(r, a, q):
    rp = r % q
    r0 = reduce_mod_pm(rp, a)
    r1 = rp - r0
    if rp - r0 == q - 1:
        r1 = 0
        r0 = r0 - 1
    else:
        r1 = (rp - r0) // a
    return r1, r0


def high_bits(r, a, q):
    r1, _ = decompose(r, a, q)
    return r1


def low_bits(r, a, q):
    _, r0 = decompose(r, a, q)
```

```
    return r0


def make_hint(z, r, a, q):
    r1 = high_bits(r, a, q)
    v1 = high_bits(r + z, a, q)
    return int(r1 != v1)


def make_hint_optimised(z0, r1, a, q):
    gamma2 = a >> 1
    if z0 <= gamma2 or z0 > (q - gamma2) or (z0 == (q - gamma2) and r1 == 0):
        return 0
    return 1


def use_hint(h, r, a, q):
    m = (q - 1) // a
    r1, r0 = decompose(r, a, q)
    if h == 1:
        if r0 > 0:
            return (r1 + 1) % m
        return (r1 - 1) % m
    return r1


def check_norm_bound(n, b, q):
    x = n % q
    x = ((q - 1) >> 1) - x
    x = x ^ (x >> 31)
    x = ((q - 1) >> 1) - x
    return x >= b


def xor_bytes(a, b):
    return bytes(a ^ b for a, b in zip(a, b))
```

Class Anel Polinomial

In [220…

```
class PolynomialRing:
    def __init__(self, q, n):
        self.q = q
        self.n = n
        self.element = Polynomial

    def gen(self):
        return self([0, 1])

    def random_element(self):
        coefficients = [random.randint(0, self.q - 1) for _ in range(self.n)]
        return self(coefficients)

    def __call__(self, coefficients):
        if isinstance(coefficients, int):
            return self.element(self, [coefficients])
        if not isinstance(coefficients, list):
            raise TypeError(
                f"Polynomials should be constructed from a list of integers, of
```

```
            )
        return self.element(self, coefficients)

    def __repr__(self):
        return f"Univariate Polynomial Ring in x over Finite Field of size {self


class Polynomial:
    def __init__(self, parent, coefficients):
        self.parent = parent
        self.coeffs = self._parse_coefficients(coefficients)

    def is_zero(self):
        return all(c == 0 for c in self.coeffs)

    def is_constant(self):
        return all(c == 0 for c in self.coeffs[1:])

    def _parse_coefficients(self, coefficients):
        l = len(coefficients)
        if l > self.parent.n:
            raise ValueError(
                f"Coefficients describe polynomial of degree greater than maximu
            )
        elif l < self.parent.n:
            coefficients = coefficients + [0 for _ in range(self.parent.n - l)]
        return coefficients

    def reduce_coefficients(self):
        self.coeffs = [c % self.parent.q for c in self.coeffs]
        return self

    def _add_mod_q(self, x, y):
        return (x + y) % self.parent.q

    def _sub_mod_q(self, x, y):
        return (x - y) % self.parent.q

    def _schoolbook_multiplication(self, other):
        n = self.parent.n
        a = self.coeffs
        b = other.coeffs
        new_coeffs = [0 for _ in range(n)]
        for i in range(n):
            for j in range(0, n - i):
                new_coeffs[i + j] += a[i] * b[j]
        for j in range(1, n):
            for i in range(n - j, n):
                new_coeffs[i + j - n] -= a[i] * b[j]
        return [c % self.parent.q for c in new_coeffs]

    def __neg__(self):
        neg_coeffs = [(-x % self.parent.q) for x in self.coeffs]
        return self.parent(neg_coeffs)

    def _add_(self, other):
        if isinstance(other, type(self)):
            new_coeffs = [
                self._add_mod_q(x, y) for x, y in zip(self.coeffs, other.coeffs)
            ]
```

```
        elif isinstance(other, int):
            new_coeffs = self.coeffs.copy()
            new_coeffs[0] = self._add_mod_q(new_coeffs[0], other)
        else:
            raise NotImplementedError("Polynomials can only be added to each oth
        return new_coeffs

    def __add__(self, other):
        new_coeffs = self._add_(other)
        return self.parent(new_coeffs)

    def __radd__(self, other):
        return self.__add__(other)

    def __iadd__(self, other):
        self = self + other
        return self

    def _sub_(self, other):
        if isinstance(other, type(self)):
            new_coeffs = [
                self._sub_mod_q(x, y) for x, y in zip(self.coeffs, other.coeffs)
            ]
        elif isinstance(other, int):
            new_coeffs = self.coeffs.copy()
            new_coeffs[0] = self._sub_mod_q(new_coeffs[0], other)
        else:
            raise NotImplementedError(
                "Polynomials can only be subtracted from each other"
            )
        return new_coeffs

    def __sub__(self, other):
        new_coeffs = self._sub_(other)
        return self.parent(new_coeffs)

    def __rsub__(self, other):
        return -self.__sub__(other)

    def __isub__(self, other):
        self = self - other
        return self

    def __mul__(self, other):
        if isinstance(other, type(self)):
            new_coeffs = self._schoolbook_multiplication(other)
        elif isinstance(other, int):
            new_coeffs = [(c * other) % self.parent.q for c in self.coeffs]
        else:
            raise NotImplementedError(
                "Polynomials can only be multiplied by each other, or scaled by
            )
        return self.parent(new_coeffs)

    def __rmul__(self, other):
        return self.__mul__(other)

    def __imul__(self, other):
        self = self * other
        return self
```

```
    def __pow__(self, n):
        if not isinstance(n, int):
            raise TypeError(
                "Exponentiation of a polynomial must be done using an integer."
            )

        # Deal with negative scalar multiplication
        if n < 0:
            raise ValueError(
                "Negative powers are not supported for elements of a Polynomial
            )
        f = self
        g = self.parent(1)
        while n > 0:
            if n % 2 == 1:
                g = g * f
            f = f * f
            n = n // 2
        return g

    def __eq__(self, other):
        if isinstance(other, type(self)):
            return self.coeffs == other.coeffs
        elif isinstance(other, int):
            if self.is_constant() and (other % self.parent.q) == self.coeffs[0]:
                return True
        return False

    def __getitem__(self, idx):
        return self.coeffs[idx]

    def __repr__(self):
        if self.is_zero():
            return "0"

        info = []
        for i, c in enumerate(self.coeffs):
            if c != 0:
                if i == 0:
                    info.append(f"{c}")
                elif i == 1:
                    if c == 1:
                        info.append("x")
                    else:
                        info.append(f"{c}*x")
                else:
                    if c == 1:
                        info.append(f"x^{i}")
                    else:
                        info.append(f"{c}*x^{i}")
        return " + ".join(info)

    def __str__(self):
        return self.__repr__()


class PolynomialRingDilithium(PolynomialRing):
    def __init__(self):
        self.q = 8380417
```

```
        self.n = 256
        self.element = PolynomialDilithium
        self.element_ntt = PolynomialDilithiumNTT

        root_of_unity = 1753
        self.ntt_zetas = [
            pow(root_of_unity, self.br(i, 8), 8380417) for i in range(256)
        ]
        self.ntt_f = pow(256, -1, 8380417)

    @staticmethod
    def br(i, k):
        bin_i = bin(i & (2**k - 1))[2:].zfill(k)
        return int(bin_i[::-1], 2)

    def sample_in_ball(self, seed, tau):
        def rejection_sample(i, xof):
            while True:
                j = xof.read(1)[0]
                if j <= i:
                    return j

        # Initialise the XOF
        xof = shake256(seed)

        # Set the first 8 bytes for the sign, and leave the rest for
        # sampling.
        sign_bytes = xof.read(8)
        sign_int = int.from_bytes(sign_bytes, "little")

        # Set the list of coeffs to be 0
        coeffs = [0 for _ in range(256)]

        # Now set tau values of coeffs to be ±1
        for i in range(256 - tau, 256):
            j = rejection_sample(i, xof)
            coeffs[i] = coeffs[j]
            coeffs[j] = 1 - 2 * (sign_int & 1)
            sign_int >>= 1

        return self(coeffs)

    def rejection_sample_ntt_poly(self, rho, i, j):
        def rejection_sample(xof):
            while True:
                j_bytes = xof.read(3)
                j = int.from_bytes(j_bytes, "little")
                j &= 0x7FFFFF
                if j < 8380417:
                    return j

        # Initialise the XOF
        seed = rho + bytes([j, i])
        xof = shake128(seed)
        coeffs = [rejection_sample(xof) for _ in range(256)]
        return self(coeffs, is_ntt=True)

    def rejection_bounded_poly(self, rho_prime, i, eta):
        def coefficient_from_half_byte(j, eta):
            """
```

```
            Rejects values until a value j < 2η is found
            """
            if eta == 2 and j < 15:
                return 2 - (j % 5)
            elif j < 9:
                assert eta == 4
                return 4 - j
            return False

        # Initialise the XOF
        seed = rho_prime + int.to_bytes(i, 2, "little")
        xof = shake256(seed)

        # Sample bytes for all n coeffs
        i = 0
        coeffs = [0 for _ in range(256)]
        while i < 256:
            # Consider two values for each byte (top and bottom four bits)
            j = xof.read(1)[0]

            c0 = coefficient_from_half_byte(j % 16, eta)
            if c0 is not False:
                coeffs[i] = c0
                i += 1

            c1 = coefficient_from_half_byte(j // 16, eta)
            if c1 is not False and i < 256:
                coeffs[i] = c1
                i += 1

        return self(coeffs)

    def sample_mask_polynomial(self, rho_prime, i, kappa, gamma_1):
        if gamma_1 == (1 << 17):
            bit_count = 18
            total_bytes = 576  # (256 * 18) / 8
        else:
            bit_count = 20
            total_bytes = 640  # (256 * 20) / 8

        # Initialise the XOF
        seed = rho_prime + int.to_bytes(kappa + i, 2, "little")
        xof_bytes = shake256(seed).read(total_bytes)
        r = int.from_bytes(xof_bytes, "little")
        mask = (1 << bit_count) - 1
        coeffs = [gamma_1 - ((r >> bit_count * i) & mask) for i in range(self.n)]

        return self(coeffs)

    def __bit_unpack(self, input_bytes, n_bits):
        if (len(input_bytes) * n_bits) % 8 != 0:
            raise ValueError(
                "Input bytes do not have a length compatible with the bit length
            )

        r = int.from_bytes(input_bytes, "little")
        mask = (1 << n_bits) - 1
        return [(r >> n_bits * i) & mask for i in range(self.n)]

    def bit_unpack_t0(self, input_bytes):
```

```
        altered_coeffs = self.__bit_unpack(input_bytes, 13)
        coefficients = [(1 << 12) - c for c in altered_coeffs]
        return self(coefficients)

    def bit_unpack_t1(self, input_bytes):
        coefficients = self.__bit_unpack(input_bytes, 10)
        return self(coefficients)

    def bit_unpack_s(self, input_bytes, eta):
        # Level 2 and 5 parameter set
        if eta == 2:
            altered_coeffs = self.__bit_unpack(input_bytes, 3)
        # Level 3 parameter set
        else:
            assert eta == 4, f"Expected eta to be either 2 or 4, got {eta = }"
            altered_coeffs = self.__bit_unpack(input_bytes, 4)

        coefficients = [eta - c for c in altered_coeffs]
        return self(coefficients)

    def bit_unpack_w(self, input_bytes, gamma_2):
        # Level 2 parameter set
        if gamma_2 == 95232:
            coefficients = self.__bit_unpack(input_bytes, 6)
        # Level 3 and 5 parameter set
        else:
            assert gamma_2 == 261888, (
                f"Expected gamma_2 to be either (q-1)/88 or (q-1)/32, got {gamma
            )
            coefficients = self.__bit_unpack(input_bytes, 4)

        return self(coefficients)

    def bit_unpack_z(self, input_bytes, gamma_1):
        # Level 2 parameter set
        if gamma_1 == (1 << 17):
            altered_coeffs = self.__bit_unpack(input_bytes, 18)
        # Level 3 and 5 parameter set
        else:
            assert gamma_1 == (1 << 19), (
                f"Expected gamma_1 to be either 2^17 or 2^19, got {gamma_1 = }"
            )
            altered_coeffs = self.__bit_unpack(input_bytes, 20)

        coefficients = [gamma_1 - c for c in altered_coeffs]
        return self(coefficients)

    def __call__(self, coefficients, is_ntt=False):
        if not is_ntt:
            element = self.element
        else:
            element = self.element_ntt

        if isinstance(coefficients, int):
            return element(self, [coefficients])
        if not isinstance(coefficients, list):
            raise TypeError(
                f"Polynomials should be constructed from a list of integers, of
            )
        return element(self, coefficients)
```

```
class PolynomialDilithium(Polynomial):
    def __init__(self, parent, coefficients):
        self.parent = parent
        self.coeffs = self._parse_coefficients(coefficients)

    def to_ntt(self):
        k, l = 0, 128
        coeffs = self.coeffs[:]
        zetas = self.parent.ntt_zetas
        while l > 0:
            start = 0
            while start < 256:
                k = k + 1
                zeta = zetas[k]
                for j in range(start, start + l):
                    t = zeta * coeffs[j + l]
                    coeffs[j + l] = coeffs[j] - t
                    coeffs[j] = coeffs[j] + t
                start = l + (j + 1)
            l >>= 1

        for j in range(256):
            coeffs[j] = coeffs[j] % 8380417

        return self.parent(coeffs, is_ntt=True)

    def from_ntt(self):
        raise TypeError(f"Polynomial is of type: {type(self)}")

    def power_2_round(self, d):
        power_2 = 1 << d
        r1_coeffs = []
        r0_coeffs = []
        for c in self.coeffs:
            r = c % self.parent.q
            r0 = reduce_mod_pm(r, power_2)
            r1_coeffs.append((r - r0) >> d)
            r0_coeffs.append(r0)

        r1_poly = self.parent(r1_coeffs)
        r0_poly = self.parent(r0_coeffs)

        return r1_poly, r0_poly

    def high_bits(self, alpha, is_ntt=False):
        coeffs = [high_bits(c, alpha, self.parent.q) for c in self.coeffs]
        return self.parent(coeffs, is_ntt=is_ntt)

    def low_bits(self, alpha, is_ntt=False):
        coeffs = [low_bits(c, alpha, self.parent.q) for c in self.coeffs]
        return self.parent(coeffs, is_ntt=is_ntt)
    def decompose(self, alpha):
        coeff_high = []
        coeff_low = []
        for c in self.coeffs:
            r1, r0 = decompose(c, alpha, self.parent.q)
            coeff_high.append(r1)
            coeff_low.append(r0)
```

```
        return self.parent(coeff_high), self.parent(coeff_low)

    def check_norm_bound(self, bound):
        return any(check_norm_bound(c, bound, self.parent.q) for c in self.coeff
    @staticmethod
    def __bit_pack(coeffs, n_bits, n_bytes):
        r = 0
        for c in reversed(coeffs):
            r <<= n_bits
            r |= c
        return r.to_bytes(n_bytes, "little")

    def bit_pack_t0(self):
        # 416 = 256 * 13 // 8
        altered_coeffs = [(1 << 12) - c for c in self.coeffs]
        return self.__bit_pack(altered_coeffs, 13, 416)

    def bit_pack_t1(self):
        # 320 = 256 * 10 // 8
        return self.__bit_pack(self.coeffs, 10, 320)

    def bit_pack_s(self, eta):
        altered_coeffs = [self._sub_mod_q(eta, c) for c in self.coeffs]
        # Level 2 and 5 parameter set
        if eta == 2:
            return self.__bit_pack(altered_coeffs, 3, 96)
        # Level 3 parameter set
        assert eta == 4, f"Expected eta to be either 2 or 4, got {eta = }"
        return self.__bit_pack(altered_coeffs, 4, 128)

    def bit_pack_w(self, gamma_2):
        # Level 2 parameter set
        if gamma_2 == 95232:
            return self.__bit_pack(self.coeffs, 6, 192)
        # Level 3 and 5 parameter set
        assert gamma_2 == 261888, (
            f"Expected gamma_2 to be either (q-1)/88 or (q-1)/32, got {gamma_2 =
        )
        return self.__bit_pack(self.coeffs, 4, 128)

    def bit_pack_z(self, gamma_1):
        altered_coeffs = [self._sub_mod_q(gamma_1, c) for c in self.coeffs]
        # Level 2 parameter set
        if gamma_1 == (1 << 17):
            return self.__bit_pack(altered_coeffs, 18, 576)
        # Level 3 and 5 parameter set
        assert gamma_1 == (1 << 19), (
            f"Expected gamma_1 to be either 2^17 or 2^19, got: {gamma_1 = }"
        )
        return self.__bit_pack(altered_coeffs, 20, 640)

    def make_hint(self, other, alpha):
        coeffs = [
            make_hint(r, z, alpha, 8380417) for r, z in zip(self.coeffs, other.c
        ]
        return self.parent(coeffs)

    def make_hint_optimised(self, other, alpha):
        coeffs = [
            make_hint_optimised(r, z, alpha, 8380417)
```

```
            for r, z in zip(self.coeffs, other.coeffs)
        ]
        return self.parent(coeffs)

    def use_hint(self, other, alpha):
        coeffs = [
            use_hint(h, r, alpha, 8380417) for h, r in zip(self.coeffs, other.co
        ]
        return self.parent(coeffs)


class PolynomialDilithiumNTT(PolynomialDilithium):
    def __init__(self, parent, coefficients):
        self.parent = parent
        self.coeffs = self._parse_coefficients(coefficients)

    def to_ntt(self):
        raise TypeError(f"Polynomial is of type: {type(self)}")

    def from_ntt(self):
        l, k = 1, 256
        coeffs = self.coeffs[:]
        zetas = self.parent.ntt_zetas
        while l < 256:
            start = 0
            while start < 256:
                k = k - 1
                zeta = -zetas[k]
                for j in range(start, start + l):
                    t = coeffs[j]
                    coeffs[j] = t + coeffs[j + l]
                    coeffs[j + l] = t - coeffs[j + l]
                    coeffs[j + l] = zeta * coeffs[j + l]
                start = j + l + 1
            l = l << 1

        for j in range(256):
            coeffs[j] = (coeffs[j] * self.parent.ntt_f) % 8380417

        return self.parent(coeffs, is_ntt=False)

    def ntt_coefficient_multiplication(self, f_coeffs, g_coeffs):
        return [(c1 * c2) % 8380417 for c1, c2 in zip(f_coeffs, g_coeffs)]

    def ntt_multiplication(self, other):
        if not isinstance(other, type(self)):
            raise ValueError

        new_coeffs = self.ntt_coefficient_multiplication(self.coeffs, other.coef
        return new_coeffs

    def __add__(self, other):
        new_coeffs = self._add_(other)
        return self.parent(new_coeffs, is_ntt=True)

    def __sub__(self, other):
        new_coeffs = self._sub_(other)
        return self.parent(new_coeffs, is_ntt=True)

    def __mul__(self, other):
```

```
        if isinstance(other, type(self)):
            new_coeffs = self.ntt_multiplication(other)
        elif isinstance(other, int):
            new_coeffs = [(c * other) % 8380417 for c in self.coeffs]
        else:
            raise NotImplementedError(
                f"Polynomials can only be multiplied by each other, or scaled by
            )
        return self.parent(new_coeffs, is_ntt=True)
```

Class com outras Propriedades Matemáticas (Matrizes, Bit Manipulation, Hints...)

In [221…

```
class Module:
    def __init__(self, ring):
        self.ring = ring
        self.matrix = Matrix

    def random_element(self, m, n):
        elements = [[self.ring.random_element() for _ in range(n)] for _ in rang
        return self(elements)

    def __repr__(self):
        return f"Module over the commutative ring: {self.ring}"

    def __str__(self):
        return f"Module over the commutative ring: {self.ring}"

    def __call__(self, matrix_elements, transpose=False):
        if not isinstance(matrix_elements, list):
            raise TypeError(
                "elements of a module are matrices, built from elements of the b
            )

        if isinstance(matrix_elements[0], list):
            for element_list in matrix_elements:
                if not all(isinstance(aij, self.ring.element) for aij in element
                    raise TypeError(
                        f"All elements of the matrix must be elements of the rin
                    )
            return self.matrix(self, matrix_elements, transpose=transpose)

        elif isinstance(matrix_elements[0], self.ring.element):
            if not all(isinstance(aij, self.ring.element) for aij in matrix_elem
                raise TypeError(
                    f"All elements of the matrix must be elements of the ring: {
                )
            return self.matrix(self, [matrix_elements], transpose=transpose)

        else:
            raise TypeError(
                "elements of a module are matrices, built from elements of the b
            )

    def vector(self, elements):
        return self.matrix(self, [elements], transpose=True)
```

```
class Matrix:
    def __init__(self, parent, matrix_data, transpose=False):
        self.parent = parent
        self._data = matrix_data
        self._transpose = transpose
        if not self._check_dimensions():
            raise ValueError("Inconsistent row lengths in matrix")

    def dim(self):
        if not self._transpose:
            return len(self._data), len(self._data[0])
        else:
            return len(self._data[0]), len(self._data)

    def _check_dimensions(self):
        return len(set(map(len, self._data))) == 1

    def transpose(self):
        return self.parent(self._data, not self._transpose)

    def transpose_self(self):
        self._transpose = not self._transpose
        return

    T = property(transpose)

    def reduce_coefficients(self):
        for row in self._data:
            for ele in row:
                ele.reduce_coefficients()
        return self

    def __getitem__(self, idx):
        assert isinstance(idx, tuple) and len(idx) == 2, "Can't access individua
        if not self._transpose:
            return self._data[idx[0]][idx[1]]
        else:
            return self._data[idx[1]][idx[0]]

    def __eq__(self, other):
        if self.dim() != other.dim():
            return False
        m, n = self.dim()
        return all([self[i, j] == other[i, j] for i in range(m) for j in range(n

    def __neg__(self):
        m, n = self.dim()
        return self.parent(
            [[-self[i, j] for j in range(n)] for i in range(m)],
            self._transpose,
        )

    def __add__(self, other):
        if not isinstance(other, type(self)):
            raise TypeError("Can only add matrices to other matrices")
        if self.parent != other.parent:
            raise TypeError("Matrices must have the same base ring")
        if self.dim() != other.dim():
            raise ValueError("Matrices are not of the same dimensions")
```

```
        m, n = self.dim()
        return self.parent(
            [[self[i, j] + other[i, j] for j in range(n)] for i in range(m)],
            False,
        )

    def __iadd__(self, other):
        self = self + other
        return self

    def __sub__(self, other):
        if not isinstance(other, type(self)):
            raise TypeError("Can only add matrices to other matrices")
        if self.parent != other.parent:
            raise TypeError("Matrices must have the same base ring")
        if self.dim() != other.dim():
            raise ValueError("Matrices are not of the same dimensions")

        m, n = self.dim()
        return self.parent(
            [[self[i, j] - other[i, j] for j in range(n)] for i in range(m)],
            False,
        )

    def __isub__(self, other):
        self = self - other
        return self

    def __matmul__(self, other):
        if not isinstance(other, type(self)):
            raise TypeError("Can only multiply matrcies with other matrices")
        if self.parent != other.parent:
            raise TypeError("Matrices must have the same base ring")

        m, n = self.dim()
        n_, l = other.dim()
        if not n == n_:
            raise ValueError("Matrices are of incompatible dimensions")

        return self.parent(
            [
                [sum(self[i, k] * other[k, j] for k in range(n)) for j in range(
                for i in range(m)
            ]
        )

    def scale(self, other):
        if not (isinstance(other, self.parent.ring.element) or isinstance(other,
            raise TypeError("Can only multiply elements with polynomials or inte

        matrix = [[other * ele for ele in row] for row in self._data]
        return self.parent(matrix, transpose=self._transpose)

    def dot(self, other):
        if not isinstance(other, type(self)):
            raise TypeError("Can only perform dot product with other matrices")
        res = self.T @ other
        assert res.dim() == (1, 1)
        return res[0, 0]
```

```
    def __repr__(self):
        m, n = self.dim()

        if m == 1:
            return str(self._data[0])

        max_col_width = [max(len(str(self[i, j])) for i in range(m)) for j in ra
        info = "]\n[".join(
            [
                ", ".join([f"{str(self[i, j]):>{max_col_width[j]}}" for j in ran
                for i in range(m)
            ]
        )
        return f"[{info}]"


class ModuleDilithium(Module):
    def __init__(self):
        self.ring = PolynomialRingDilithium()
        self.matrix = MatrixDilithium

    def __bit_unpack(self, input_bytes, m, n, alg, packed_len, *args):
        poly_bytes = [
            input_bytes[i : i + packed_len]
            for i in range(0, len(input_bytes), packed_len)
        ]
        matrix = [
            [alg(poly_bytes[n * i + j], *args) for j in range(n)] for i in range
        ]
        return self(matrix)

    def bit_unpack_t0(self, input_bytes, m, n):
        packed_len = 416
        algorithm = self.ring.bit_unpack_t0
        return self.__bit_unpack(input_bytes, m, n, algorithm, packed_len)

    def bit_unpack_t1(self, input_bytes, m, n):
        packed_len = 320
        algorithm = self.ring.bit_unpack_t1
        return self.__bit_unpack(input_bytes, m, n, algorithm, packed_len)

    def bit_unpack_s(self, input_bytes, m, n, eta):
        # Level 2 and 5 parameter set
        if eta == 2:
            packed_len = 96
        # Level 3 parameter set
        elif eta == 4:
            packed_len = 128
        else:
            raise ValueError("Expected eta to be either 2 or 4")
        algorithm = self.ring.bit_unpack_s
        return self.__bit_unpack(input_bytes, m, n, algorithm, packed_len, eta)

    def bit_unpack_w(self, input_bytes, m, n, gamma_2):
        # Level 2 parameter set
        if gamma_2 == 95232:
            packed_len = 192
        # Level 3 and 5 parameter set
        elif gamma_2 == 261888:
            packed_len = 128
```

```
        else:
            raise ValueError("Expected gamma_2 to be either (q-1)/88 or (q-1)/32
        algorithm = self.ring.bit_unpack_w
        return self.__bit_unpack(input_bytes, m, n, algorithm, packed_len, gamma

    def bit_unpack_z(self, input_bytes, m, n, gamma_1):
        # Level 2 parameter set
        if gamma_1 == (1 << 17):
            packed_len = 576
        # Level 3 and 5 parameter set
        elif gamma_1 == (1 << 19):
            packed_len = 640
        else:
            raise ValueError("Expected gamma_1 to be either 2^17 or 2^19")
        algorithm = self.ring.bit_unpack_z
        return self.__bit_unpack(input_bytes, m, n, algorithm, packed_len, gamma


class MatrixDilithium(Matrix):
    def __init__(self, parent, matrix_data, transpose=False):
        super().__init__(parent, matrix_data, transpose=transpose)

    def check_norm_bound(self, bound):
        for row in self._data:
            if any(p.check_norm_bound(bound) for p in row):
                return True
        return False

    def power_2_round(self, d):
        m, n = self.dim()

        m1_elements = [[0 for _ in range(n)] for _ in range(m)]
        m0_elements = [[0 for _ in range(n)] for _ in range(m)]

        for i in range(m):
            for j in range(n):
                m1_ele, m0_ele = self[i, j].power_2_round(d)
                m1_elements[i][j] = m1_ele
                m0_elements[i][j] = m0_ele

        return self.parent(m1_elements, transpose=self._transpose), self.parent(
            m0_elements, transpose=self._transpose
        )

    def decompose(self, alpha):
        m, n = self.dim()

        m1_elements = [[0 for _ in range(n)] for _ in range(m)]
        m0_elements = [[0 for _ in range(n)] for _ in range(m)]

        for i in range(m):
            for j in range(n):
                m1_ele, m0_ele = self[i, j].decompose(alpha)
                m1_elements[i][j] = m1_ele
                m0_elements[i][j] = m0_ele

        return self.parent(m1_elements, transpose=self._transpose), self.parent(
            m0_elements, transpose=self._transpose
        )
```

```
    def __bit_pack(self, algorithm, *args):
        return b"".join(algorithm(poly, *args) for row in self._data for poly in

    def bit_pack_t1(self):
        algorithm = self.parent.ring.element.bit_pack_t1
        return self.__bit_pack(algorithm)

    def bit_pack_t0(self):
        algorithm = self.parent.ring.element.bit_pack_t0
        return self.__bit_pack(algorithm)

    def bit_pack_s(self, eta):
        algorithm = self.parent.ring.element.bit_pack_s
        return self.__bit_pack(algorithm, eta)

    def bit_pack_w(self, gamma_2):
        algorithm = self.parent.ring.element.bit_pack_w
        return self.__bit_pack(algorithm, gamma_2)

    def bit_pack_z(self, gamma_1):
        algorithm = self.parent.ring.element.bit_pack_z
        return self.__bit_pack(algorithm, gamma_1)

    def to_ntt(self):
        data = [[x.to_ntt() for x in row] for row in self._data]
        return self.parent(data, transpose=self._transpose)

    def from_ntt(self):
        data = [[x.from_ntt() for x in row] for row in self._data]
        return self.parent(data, transpose=self._transpose)

    def high_bits(self, alpha, is_ntt=False):
        matrix = [
            [ele.high_bits(alpha, is_ntt=is_ntt) for ele in row] for row in self
        ]
        return self.parent(matrix)

    def low_bits(self, alpha, is_ntt=False):
        matrix = [
            [ele.low_bits(alpha, is_ntt=is_ntt) for ele in row] for row in self.
        ]
        return self.parent(matrix)

    def make_hint(self, other, alpha):
        matrix = [
            [p.make_hint(q, alpha) for p, q in zip(r1, r2)]
            for r1, r2 in zip(self._data, other._data)
        ]
        return self.parent(matrix)

    def make_hint_optimised(self, other, alpha):
        matrix = [
            [p.make_hint_optimised(q, alpha) for p, q in zip(r1, r2)]
            for r1, r2 in zip(self._data, other._data)
        ]
        return self.parent(matrix)

    def use_hint(self, other, alpha):
        matrix = [
            [p.use_hint(q, alpha) for p, q in zip(r1, r2)]
```

```
            for r1, r2 in zip(self._data, other._data)
        ]
        return self.parent(matrix)

    def sum_hint(self):
        return sum(c for row in self._data for p in row for c in p)
```

Implementação do Dilithium

Definição dos Paramêtros

$d = 13, k = 8, \ell = 7, \eta = 2, \tau = 60, \omega = 75, \gamma_1 = 2^{19} = 524288, \gamma_2 = \frac{q-1}{32} = 261888$

In [222…

```
d = 13
k = 8
l = 7
eta = 2
tau = 60
omega = 75
gamma_1 = 524288 # 2^19
gamma_2 = 261888  # (q-1)/32
beta = tau * eta

M = ModuleDilithium()
R = M.ring
random_bytes = os.urandom
```

Funções Auxiliares

`func_h`

1. Gera um hash de comprimento fixo a partir de uma entrada usando a função SHAKE256.

In [228…

```
def func_h(input_bytes, length):
    return shake256(input_bytes).read(length)
```

`expand_matrix_from_seed`

1. Expande uma seed rho para criar a matriz pública A no domínio NTT
   - Inicializa uma matriz A_data com dimensões k x l.
   - Para cada posição (i, j), chama `rejection_sample_ntt_poly` para gerar um polinômio com coeficientes uniformemente distribuídos no domínio NTT.
   - Retorna a matriz como um objeto do tipo M (Module).

In [ ]:

```
def expand_matrix_from_seed(rho):
    A_data = [[0 for _ in range(l)] for _ in range(k)]
    for i in range(k):
        for j in range(l):
            A_data[i][j] = R.rejection_sample_ntt_poly(rho, i, j)
    return M(A_data)
```

`expand_vector_from_seed`

1. Expande uma semente ρ' para produzir os vetores secretos s1 e s2.
    - Gera l polinômios para s1 e k polinômios para s2 usando rejection_bounded_poly.
    - Cada polinômio tem coeficientes limitados pelo parâmetro eta (pequenos valores para segurança).
    - Retorna s1 e s2 como vetores do tipo M.vector.

In [ ]:
```
def expand_vector_from_seed(rho_prime):
    s1_elements = [
        R.rejection_bounded_poly(rho_prime, i, eta) for i in range(l)
    ]
    s2_elements = [
        R.rejection_bounded_poly(rho_prime, i, eta)
        for i in range(l, l + k)
    ]
    s1 = M.vector(s1_elements)
    s2 = M.vector(s2_elements)
    return s1, s2
```

`expand_mask_vector`

1. Gera o vetor máscara y usado no processo de assinatura.
    - Cria l polinômios usando sample_mask_polynomial, com coeficientes em um intervalo definido por gamma_1;
    - Usa rho_prime como semente e kappa como um contador para garantir unicidade;
    - Retorna o vetor y como um objeto M.vector.

In [ ]:
```
def expand_mask_vector(rho_prime, kappa):
    elements = [
        R.sample_mask_polynomial(rho_prime, i, kappa, gamma_1)
        for i in range(l)
    ]
    return M.vector(elements)
```

`pack_pk`

1. Empacota a chave pública em um formato compacto.
    - Concatena a semente rho (32 bytes) com a representação empacotada de t1 (obtida via bit_pack_t1);
    - t1 contém os bits altos do vetor t, comprimidos para eficiência.

In [ ]:
```
def pack_pk(rho, t1):
    return rho + t1.bit_pack_t1()
```

`pack_sk`

1. Empacota a chave privada em um formato compacto e seguro.

    - ρ (semente da matriz A, 32 bytes),
    - κ (semente auxiliar, 32 bytes),
    - tr (hash da chave pública, 32 bytes),
    - Bytes empacotados de $s1$ e $s2$ (via `bit_pack_s`),
    - Bytes empacotados de t0 (via `bit_pack_t0`).

In [ ]:
```
def pack_sk(rho, K, tr, s1, s2, t0):
    s1_bytes = s1.bit_pack_s(eta)
    s2_bytes = s2.bit_pack_s(eta)
    t0_bytes = t0.bit_pack_t0()
    return rho + K + tr + s1_bytes + s2_bytes + t0_bytes
```

`pack_h`

1. Empacota as dicas h (hint) de forma eficiente para reduzir o tamanho da assinatura.

    - Extrai as posições dos coeficientes não-zero (1s) de cada polinômio em h;
    - Armazena essas posições em uma lista `packed` e calcula os offsets para reconstrução;
    - Preenche com zeros até atingir o comprimento ω e retorna como bytes.

In [ ]:
```
def pack_h(h):
    non_zero_positions = [
        [i for i, c in enumerate(poly.coeffs) if c == 1]
        for row in h._data
        for poly in row
    ]
    packed = []
    offsets = []
    for positions in non_zero_positions:
        packed.extend(positions)
        offsets.append(len(packed))
    padding_len = omega - offsets[-1]
    packed.extend([0 for _ in range(padding_len)])
    return bytes(packed + offsets)
```

`pack_sig`

1. Empacota a assinatura em um formato compacto.

    - ζ (desafio hash, 32 bytes);
    - Bytes empacotados de z (via `bit_pack_z`);
    - Os bytes de z, empacotados com z.bit_pack_z(gamma_1).
    - Os bytes de h, empacotados com pack_h(h).

In [ ]:
```
def pack_sig(c_tilde, z, h):
    return c_tilde + z.bit_pack_z(gamma_1) + pack_h(h)
```

`unpack_pk`

1. Extrair a chave pública a partir de sua representação em bytes.

    - Divide pk_bytes em duas partes:
        - ρ: Primeiros 32 bytes, uma semente usada para gerar a matriz A.
        - $t1\_bytes$: Bytes restantes, que contêm o vetor t1 empacotado.
    - Desempacota t1 usando M.bit_unpack_t1(t1_bytes, k, 1), retornando-o como um vetor do tipo M.

In [ ]:
```
def unpack_pk(pk_bytes):
    rho, t1_bytes = pk_bytes[:32], pk_bytes[32:]
    t1 = M.bit_unpack_t1(t1_bytes, k, 1)
    return rho, t1
```

`unpack_sk`

1. Desempacotar a chave privada a partir de seus bytes.

    - Verifica se o comprimento de sk_bytes é consistente com os parâmetros η, k e l;
    - Divide sk_bytes em:
        - ρ, κ, tr: 32 bytes cada, representando semente, chave de mascaramento e hash da chave pública, respectivamente.
        - Bytes de $s1$, $s2$ e $t0$, que são vetores de polinômios;
    - Desempacota:
        - $s1$ com M.bit_unpack_s($s1_bytes$, l, 1, η)
        - $s2$ com M.bit_unpack_s($s2_bytes$, k, 1, η)
        - $t0$ com M.bit_unpack_s($t0_bytes$, l, 1, 1)

In [ ]:
```python
def unpack_sk(sk_bytes):
    if eta == 2:
        s_bytes = 96
    else:
        s_bytes = 128
    s1_len = s_bytes * l
    s2_len = s_bytes * k
    t0_len = 416 * k
    if len(sk_bytes) != 3 * 32 + s1_len + s2_len + t0_len:
        raise ValueError("SK packed bytes is of the wrong length")
    # Split bytes between seeds and vectors
    sk_seed_bytes, sk_vec_bytes = sk_bytes[:96], sk_bytes[96:]
    # Unpack seed bytes
    rho, K, tr = (
        sk_seed_bytes[:32],
        sk_seed_bytes[32:64],
        sk_seed_bytes[64:96],
    )
    # Unpack vector bytes
    s1_bytes = sk_vec_bytes[:s1_len]
    s2_bytes = sk_vec_bytes[s1_len : s1_len + s2_len]
    t0_bytes = sk_vec_bytes[-t0_len:]
    # Unpack bytes to vectors
    s1 = M.bit_unpack_s(s1_bytes, l, 1, eta)
    s2 = M.bit_unpack_s(s2_bytes, k, 1, eta)
    t0 = M.bit_unpack_t0(t0_bytes, k, 1)
    return rho, K, tr, s1, s2, t0
```

`unpack_h`

1. Reconstruir a matriz de dicas h a partir de seus bytes.
   - Usa os últimos k bytes de h_bytes como offsets para identificar posições;
   - Reconstrói as posições não-zero dos polinômios a partir dos bytes anteriores.
   - Gera uma matriz de polinômios com coeficientes 0 ou 1 nas posições indicadas, retornando-a como um objeto M.

In [ ]:
```python
def unpack_h(h_bytes):
    offsets = [0] + list(h_bytes[-k :])
    non_zero_positions = [
        list(h_bytes[offsets[i] : offsets[i + 1]]) for i in range(k)
    ]
    matrix = []
    for poly_non_zero in non_zero_positions:
        coeffs = [0 for _ in range(256)]
        for non_zero in poly_non_zero:
            coeffs[non_zero] = 1
        matrix.append([R(coeffs)])
    return M(matrix)
```

`unpack_sig`

1. Desempacotar os componentes da assinatura a partir de seus bytes.
   - Divide sig_bytes em:
     - ζ: Primeiros 32 bytes, o desafio hash.
     - z_bytes: Bytes intermediários até os últimos $k + \omega$, contendo z empacotado.

- h_bytes: Últimos $k + \omega$ bytes, contendo h empacotado.

- Desempacota:
  - z com M.bit_unpack_z(z_bytes, l, 1, γ_1);
  - h com unpack_h(h_bytes).

In [ ]:
```
def unpack_sig(sig_bytes):
    c_tilde = sig_bytes[:32]
    z_bytes = sig_bytes[32 : -(k + omega)]
    h_bytes = sig_bytes[-(k + omega) :]
    z = M.bit_unpack_z(z_bytes, l, 1, gamma_1)
    h = unpack_h(h_bytes)
    return c_tilde, z, h
```

Implementação do Dilithium

A função `keygen` é responsável por gerar o par de chaves pública e privada:

1. Gera uma semente aleatória ζ usando os.urandom;
2. Expande ζ com SHAKE-256 para obter ρ, ρ' e K;
3. Gera a matriz $\mathbf{A}$ no domínio NTT a partir de ρ usando _expand_matrix_from_seed;
4. Gera os vetores secretos $\mathbf{s}_1$ e $\mathbf{s}_2$ com coeficientes em $[-\eta, \eta]$ a partir de ρ';
5. Computa $\mathbf{t} = \mathbf{A}\mathbf{s}_1 + \mathbf{s}_2$, com $\mathbf{s}_1$ no domínio NTT, e aplica power_2_round para obter $\mathbf{t}_1$ e $\mathbf{t}_0$;
6. Empacota a chave pública como $(\rho, \mathbf{t}_1)$ e a chave secreta como $(\rho, K, tr, \mathbf{s}_1, \mathbf{s}_2, \mathbf{t}_0)$, onde $tr = H(\rho \| \mathbf{t}_1)$.

In [224…
```
def keygen():
    #1
    # Gerar Zeta Aleatorio (32 bytes)
    zeta = random_bytes(32)

    #2
    # Gerar a seed com Zeta
    seed_bytes = func_h(zeta, 128)
    # Separar os bytes da seed em rho, rho_prime e K
    rho, rho_prime, K = seed_bytes[:32], seed_bytes[32:96], seed_bytes[96:]

    #3
    # Gerar a matriz A ∈ R^(k x l)
    A_hat = expand_matrix_from_seed(rho)

    #4
    # Gerar os vetores de erro s1 ∈ R^l, s2 ∈ R^k
    s1, s2 = expand_vector_from_seed(rho_prime)
    s1_hat = s1.to_ntt()

    #5
    # Multiplicação de matrizes
    t = (A_hat @ s1_hat).from_ntt() + s2
    t1, t0 = t.power_2_round(d)

    #6
    # Empacotar os bytes
```

```
    pk = pack_pk(rho, t1)
    tr = func_h(pk, 32)
    sk = pack_sk(rho, K, tr, s1, s2, t0)
    return pk, sk
```

A função `sign` gera uma assinatura para uma mensagem M:

1. Desempacota a chave secreta e regenera $\mathbf{A}$ a partir de ρ;
2. Computa $\mu = H(tr \| M)$ e $\rho' = H(K \| \mu)$ para a versão determinística;
3. Gera o vetor de mascaramento $\mathbf{y}$ com _expand_mask_vector usando ρ' e um nonce κ;
4. Computa $\mathbf{w} = \mathbf{A}\mathbf{y}$ no domínio NTT e decompõe em $\mathbf{w}_1$ e $\mathbf{w}_0$ com decompose;
5. Gera o desafio c como $H(\mu \| \mathbf{w}_1)$ e computa $\mathbf{z} = \mathbf{y} + c\mathbf{s}_1$;
6. Aplica verificações de rejeição (e.g., $\|\mathbf{z}\|_\infty < \gamma_1 - \beta$) e calcula as dicas $\mathbf{h}$ com `make_hint_optimised`;
7. Empacota a assinatura como $(\hat{c}, \mathbf{z}, \mathbf{h})$. O processo está em conformidade com a especificação.

In [225…

```
def sign(sk_bytes, m):
    #1
    # Desampacota a sk e regenera a matriz A
    rho, K, tr, s1, s2, t0 = unpack_sk(sk_bytes)
    # Gerar a matriz A ∈ R^(k x l)
    A_hat = expand_matrix_from_seed(rho)

    #2
    # Gera Mu, Kappa e rho_prime
    mu = func_h(tr + m, 64)
    kappa = 0
    rho_prime = func_h(K + mu, 64)

    #3
    # Gerar o Vetor Máscara y ∈ R^l
    s1 = s1.to_ntt()
    s2 = s2.to_ntt()
    t0 = t0.to_ntt()
    alpha = gamma_2 << 1
    while True:
        y = expand_mask_vector(rho_prime, kappa)
        y_hat = y.to_ntt()
        kappa += l #nonce

        #4
        #w é o resultado da multiplicação da matriz A por y
        #Converte de volta do NTT para o domínio normal
        w = (A_hat @ y_hat).from_ntt()

        # Decompõe w em w1 (bits altos) e w0 (bits baixos) usando alpha
        w1, w0 = w.decompose(alpha)
        # Empacota w1 para o hash do desafio
        w1_bytes = w1.bit_pack_w(gamma_2)

        #5
        #c é um polinômio esparso com coeficientes em {-1, 0, 1}
        #Calculado a partir de Mu e w1, garantindo ligação com a mensagem
        c_tilde = func_h(mu + w1_bytes, 32)
```

```
        c = R.sample_in_ball(c_tilde, tau)
        c = c.to_ntt()

        #6
        # Calcular z = y + c * s1
        #z é a parte principal da assinatura
        #Adiciona o desafio c escalado por s1 a y
        z = y + (s1.scale(c)).from_ntt()

        # Verifica se z está dentro do limite de norma (gamma_1 - beta)
        # Se exceder, rejeita e tenta novamente
        if z.check_norm_bound(gamma_1 - beta):
            continue

        # Calcula w0 - c * s2 e verifica a norma
        # Garante que a assinatura não revele informações sobre s2
        w0_minus_cs2 = w0 - s2.scale(c).from_ntt()
        if w0_minus_cs2.check_norm_bound(gamma_2 - beta):
            continue

        # Calcula c * t0 e verifica a norma
        # t0 é a parte baixa de t, e sua norma também deve ser limitada
        c_t0 = t0.scale(c).from_ntt()
        if c_t0.check_norm_bound(gamma_2):
            continue

        # Calcula w0 - c * s2 + c * t0 para gerar as dicas (hints)
        # As dicas h ajudam a reconstruir w1 na verificação
        w0_minus_cs2_plus_ct0 = w0_minus_cs2 + c_t0
        h = w0_minus_cs2_plus_ct0.make_hint_optimised(w1, alpha)

        # Verifica se o número de dicas não excede omega
        # Limite imposto para eficiência e segurança
        if h.sum_hint() > omega:
            continue

        # 7
        # Se todas as verificações passarem, empacota e retorna a assinatura
        # A assinatura contém c_tilde, z e h
        return pack_sig(c_tilde, z, h)
```

A função `verify` verifica a validade de uma assinatura:

1. Desempacota a chave pública $(\rho, \mathbf{t}_1)$ e a assinatura $(\hat{c}, \mathbf{z}, \mathbf{h})$.
2. Verifica se o número de dicas em $\mathbf{h}$ é $\leq \omega$ e se $\|\mathbf{z}\|_\infty < \gamma_1 - \beta$.
3. Reconstrói $\mathbf{A}$ a partir de ρ e computa $\mathbf{w}'_1 = \text{UseHint}(\mathbf{h}, \mathbf{Az} - c\mathbf{t}_1 \cdot 2^d, 2\gamma_2)$.
4. Confirma que $\hat{c} = H(\mu \| \mathbf{w}'_1)$.

In [226…

```
def verify(pk_bytes, m, sig_bytes):
    #1
    # Desempacota a pk e a assinatura
    rho, t1 = unpack_pk(pk_bytes)
    c_tilde, z, h = unpack_sig(sig_bytes)

    #2
    #Verificar se o comprimento de h é válido
    if h.sum_hint() > omega:
        return False
```

```
    # Verificar se z está dentro dos limites
    if z.check_norm_bound(gamma_1 - beta):
        return False

    #3.1
    #Reconstruir
    A_hat = expand_matrix_from_seed(rho)
    tr = func_h(pk_bytes, 32)
    mu = func_h(tr + m, 64)
    c = R.sample_in_ball(c_tilde, tau)

    #3.2
    # w1 = h.make_hint_optimised
    c = c.to_ntt()
    z = z.to_ntt()
    t1 = t1.scale(1 << d)
    t1 = t1.to_ntt()
    Az_minus_ct1 = (A_hat @ z) - t1.scale(c)
    Az_minus_ct1 = Az_minus_ct1.from_ntt()
    w_prime = h.use_hint(Az_minus_ct1, 2 * gamma_2)
    w_prime_bytes = w_prime.bit_pack_w(gamma_2)

    #4
    # Confirmar se c_tilde é igual a func_h(mu + w_prime_bytes, 32)
    return c_tilde == func_h(mu + w_prime_bytes, 32)
```

Run

In [227…

```
print ("======= Dilithium ======= \n")
pk, sk = keygen()
print(f"Chave Pública: {pk.hex()}")
print(f"Chave Privada: {sk.hex()}")
msg = b"Mensagem em Dilithium"

print(f"\nMensagem: {msg}")
sig = sign(sk, msg)
print(f"\nAssinatura: {sig.hex()}")

print("\nVerificação?", verify(pk, msg, sig))
```

======= Dilithium =======

Chave Pública: 4130977e9fb76947be0b439925443004fc3adbc22a5624ae46c69daa33d7f33a96
63124433825d4d155e046e9ca4e4d31c280bb77d36f544f228602fda1fdd3be5c7d6df20e718bda04
0ccfcb4ef19cdb41accb36d647b249c9a8b777fd023e8653fdafbbaaa81d1837b98d33ac390530a38
b1a69550393c274bef514268c845d2ad200e4304b5e9da39fb4051bc0d59625d732941e12ef9ac3a8
0c0ebf2812f7cfb91d0c7be4949b9acdce9d4658b111ade58a3d142cd24e759892500fd11a6b86863
3e78cc5d3f77e552d3988d084a4973d8f98f56ffe32249fce8775fdcfa0302a3c4dd298509157add2
eab4f454e6e26474642d45dea48d2f68111d611d35705754ff228f1a1d89094651c8b8092727b1c55
9c522ccb568ce3cda2726e9f69e167ea4653738a0a30f2ccc0920793c31127eed54aef7482fced917
d9b699e352ffd777fabd54e2f0ad4701823c41c66315fbd24a8920434a860a6fc99ff1845c915080e
90915a10b3327779547c247fd372d46aa52f8b30cef255e311d2d7fa1b27cbbf17a0401480f70379f
45d0ad5ddc62b049f306ec3e64561d6d13dceca22ac070a6492689e252c7c4a09fb52b16f7f9a0aa0
01feb1135e62ea1b0f6d218f5dc2ebc468945c3c7a995fdd653be4a64e6523873b3f4eee52187ac66
af072d0680d5802d3c5ef25210e16a922ee0143f15a4bcc1eff075358ece0f2b98d894ce54911b0a7
aaf0d9d830be5901dc74e64705f9ae4635b4f03186108acbc941aafb1cb2f6a95654907335f069c3e
1fb5366934bc130b973c3dbd579c56e91c9ab648fa5c2c4ee87e6fc77aff524d1103d4c90508c8a6d
5c7ac6e073745c4b79ec61b9491cb1b0c1c2c27395b516827741bcfbce0ffcac6cfb4dd9d394cc78c
ee30dc121fc6e1b626a9724ccd2dc587c05baa9ef3f007b6e63e4c2331cdba7c61c008be5a097a54f
3042636ade5b1169c917eba9838dac53977b8f7513bb5d490d9bb146c0afb9effbdf4f8cb18fea83b
6e35134a3e07f1542037740744221a3253bdea68569cb2f0a0528ac405bc4fef89b1f555213c300de
3c91c764ac03f23d3c39a550620b4e42782f98ad6bc79186311883046a18ac38f40349a846a6d1be7
3d25b335329813b1a43e72be59910c201fd01b50493553c381dc2eb2ae7b37128d747ce6d686af016
195f810a05e1146da8657eef4111b31fcecd0fbff6fc1f00d949675292df9056aa6df00b9f588dfa7
50b304fff0c934e6e2a87bbe683fe3825480672c6c1fa951c30e1c7d5563f46baf6e296f7199f9763
4c47a828b592d76cce413c282543eaa79b3dee1e6e7db76859395f8366eacb5fd4e000e11f0df970e
8714d393d39d3d24f396bf3e71d4cac58c694a988423ff97eb74d75fd1b9743ffd29a6dbd44f104b3
82b5ca80e8e6901b37418ade8288b9da32617aa736731757fdc1f61e5996a8b3668ecc264e2971fff
e51527a94b6d2034aafe571027b44def291d18dbc3829eb6887cb83b40257fed82092fb37c5323377
9ad41fb76f7e10c20d655c9ecb9619a161bddca3b9be27ed000a37ac68df614631bd2a20b60820fbb
68a1d08b764624e8b494e3a4e639589f5d00080887a054e8ebb1027f1460b1ae3191c76698e3c2737
8565072ffd9c60ff086624ff8372d0fb1dccf31ff3d1a956d88f4bfe95b0032b18e501c8e5a3aaa0e
9e8ed0dbd76222ec9ee639abf129e1b5222a7159445b28371c10d04d8c4bbd6e70139a5f2447975c1
d1853ee38881aa4b3aed3d7398418b4b71f289812715dbb1183551ffcda097b4588709075be441758
e7e070da43f568a43b8c5f315d103522f59d4c1795135df95a3e0c771e7069a630c240976fb2cd560
9e13a2b3f81193bb318e1457527d92a1c4ac86da7b9f266f90d1fbfc8b6112775a3fff57786fad656
35ec0a4fc9261c4164bb6e935d73fdfdcc9e779bac48f4c6d64075ed9c070270fbd7940ce06271829
58f4339bd87c8d8b7130169df01340606ad1eee9d9ad3041f80490fa723bf7530571679a5f6775ffe
baae1c66ca8ada1c607d60304fb83888e8e3c09ea01276525974b9e34920171b1f974945596b7575a
2839e76cbfaf130df81084d41d6d59481f98eca65f07c68bdef65baf18cdb58e4453bd1dd2a99baac
760143e999b8d41bec20f062a6aaa0df05d0d1efc5f48dd554eea8e69cb6c553bd15bdc3c5aa3fd29
dc2bd78c04a0e8321f71641bf4f8b1280cfc97566e6d42d5c87222e82e7e4439c0b71bf86ee3202ac
7ec466f174b2ccc1984712550bda58b90e56dfe1e345bc51a0b7b6aebd1f172e768b137b9a94d8ee8
080d144cdca3c8c5875895ce2389fee48462168de22df111b4de64d4352309373fa10c25e455104ef
1d99ee25382f5fe2c5fb9762958a6ca6f675df0f7abcec9730b8a19915f99745991931c5e04c79d0f
465c30db1537cec034c70dd4751e78f60a2ab224bd11bd46e31efc96335b23a8c30a91f808bf5117e
1c35521d1da89a6bc5ff98c860475142fddedcfcdbb076d8f5d51f7e6262e23db8ba27891e4982146
51971f26e4f703cc7a758aa7164ae8c65e27a610e19ef965548118876278abb599a43d0ceba650039
3a5d31c1b7c7ed70f93ec1e90c25640abe2491edeaec1c292e8b1681ffc53b171a3c299969ec1246f
e5857ce5c1f54ae1e1966b304c6c52d73146479e87e9c5e5d153aa15ee1ed0bfcabd8069676d193bb
7c05cfde3fce58654a966a950bdbbaf357715775f0ed1febd8966f3ae24a4eed645474bb380083f24
bba5598a4ff19835a000d3f2c695034705f740fb67da7cbf8faf87916cfcfa22f8eb6d807a26c2ca3
e706054ee5455775d645821d43803932606478346e836dc271c2a96e2765a3956ca2be2eeeef6fe7e
9157e4399d8d674e2151259dcfb032685200970895358053908ad2fdf88cad5791b5914d77d7d1623
e0e9b6b262c35bc4a31ff1708e8d94f7560e7ede798939cda0172bb561d17432f24a9ff4cb87b10bd
f5a5e8aa6aac15f27d6b4e82679c7855e6eab0d8b1ffddf4ccc772f78788f08bd8f05463792c42ceb
b510904937dc1bd3d73ef9d10a9ffad5d1bb12754c9bfe09d8b8c8b26e4e34e0382b496086a784220
38decee697e33dfa27eeeeb97cd4af2f4eac85dec3694661e3a710968b5aa55959c372ca4fdfe5264
c5348b17986be02a9ca656aa0d8c4b7cf91915b991153169d352b34a730ceaa6516dd305eb79be88f
df2dd7fac1b1ea65a6da99cef36be70cdbe357c11f2bdebac5e9bcf13c3dc4f48991883ada22f4149

95faba3d47b7519b636e846bafdbeacda6dc83408222e1e76fa42e3a10564f36d4746cdb38d024394
765878896ad6338650e1f345a92f0a2443272cbbe02454b07be3d89105b7db4702a2e3b69fc975468
5a0bfbe4faf8a3eb0745550a2e7e951afc7c39eca350b140e3d5c90d514c709b2c3f14f8f32bdfb04
596bea1c4f556af6ff782e115eee6fcee5c5540732cf1077cfcad974c0e576c7a74bff6b71a11c97f
998f2f0d8332aa046f8710caf02787664fa8f6412b6577a292885fbda7cadf9dfed76f09bbe965d2c
ee5fb3b56e9ec246af8daf8ef01e4d1143f95a0f3c2493e86142ae8a90229e94c75175a2b22d580f2
28648d76239cd8d
Chave Privada: 4130977e9fb76947be0b439925443004fc3adbc22a5624ae46c69daa33d7f33a0c
c0b1f5b64c6a9cbb8ee5fee7d45d4dfbc995bc4db6cb513646e326109acf11d21501a628ec911614c
8c5edf79dce3f50b67723afe97691400d0db7781e90f808a22001a741d2b430939060138905194726
820872090684214322 0a03885aa6290c936d42160e44b20064188040a42122180a5314020a8330230
3802110260b1548c3c2119b928c22184d00364813a420c3306de04448d8042e5c3800134012c13864
08150c62424488c8640114840c9340188470c916681c299120027102878d5ca66019354809c30ca20
88d1cb2059c068d11c848000632d9b44c49426124b1491447829c860919240a022350438200a48849
112202c332611447689b44124c465184a84ca3168c8134918122496496309042822340 4a094646a34
88da0264e0217469410451c1629e216202208648cc8855ba271811240d1442c02c86823c304432221
24204d040611e122811020820a114ee3204e1bb51123162ad1c06153260243b448891630e1062ca2c
244d08051230644e30282a3a06d99960851c2691c030010182c1295718414098bc6454a328e581044
4a3052249260634628130309532488d9920d01202848966d8b90658a080199c43119c00421c805d0b
865e0408d21116203c0508ba0650ac264e4102a50c68898b60d5a206823a38510980522b78d24a611
d91426a0a405a13406200820041070020 76460b870c4429224181013202421b80058948c22062ad91
071caa28159286c211741640446a0440c40465284c4681aa72893b011dcc04888126e4a12849bc611
e33292e4a6259c348082389258c0050b19840a4685189544894812 08a3241b12111348692000500a4
46a08b720240861421481e048028818259a802594180162a82d03414e60c804cc282218210dc0020c
6388892392415386091825680 1920d61b42053286513960542927053 4848c9228ad38805e13865480
6611b960900128000a19189b0901bc50d1ab2712239024a06265c3484243244d1b24410c065489471
0986012289401cb6056224200b48511ac470da48620aa76c0a106158348c11160d1cb40c121032d09
424d8926cd2221109496d4c2241c0b490d0b2894ca4611aa94ce0902d4888008b162c5a144a99460e
8c96090a154a883069248531e3a28102494c23094c00940062c0618838884090304c244d18402e603
604c422041b1421d4b409c2086c0c330a00a64498a00d83c24144b06d61148020956c14844ddb2464
020321db1680c2382da3044dd0c024230371e0c4008a402a53962060c0681bb684e1888003844c101
6325940650096305040605a426d8b9884a4888d1c476d1b122a8ac6104b222651044e8816009b1866
01046c81409140c669d33072420072e2422a19218600b191c11671db082601b810d1c60801121053b
21162261044b82960282e03020252488d4432229202460a136191982d228620c14620d38271191289
1a1751dc4291e494640482719a268911068d0cc4045428641c950942320548084e821869d082511b8
24120b741c4326954240e81228a98b4851ab68503c54510816820a051d1c829521671a3920459a449
22941043226520a66823105203c14cd2242619930413088da2c001d8c411132401e1b008cca848d82
48e2007304830490133252311261c928d9322919c984102802002b484a21040cbc69093c688d8a850
02082064800404312a0b024e242511a2022519312cc20241d81484c0142ee4c281a23246110770523
68852b2648aa08cd3025021c170890692ca0486d39204448824010 7660a4404d4287182000ed3488e
0142110041822394515334502023091407 4500b560183286da004a91c44d9cc891e12646db3024da9
4511b437204286e21a244db4692e4a884d40606a0c82164922d51322a53426e8a3404089420013066
0cc6046216111ba825d8248898b26c0c220508b76d9a180242906123a00458c4909026901917210aa
42dd422329444895318418420 52e3182410c48ce2c4214a3862034472a308855046221827121c450c
20080284b404032492512480da422241888412b90d01227144b850c84600ca108ac438110c342ea32
88e58046654b94ad9ce918a31240bbb4d33119416a8e225f12c88c7da9ede459c8270e97f90cd5009
2929b27adf2f18f8d70ff983f0442115f8480c8c9586eaf403d831c7f5a9b8853580cdfaf9da48059
64ded4e0742a6ef60314e718d1db6b6292d59bae25fe71bcad6e0acc429352b5e6c3251b5ffedf291
231fe0b5a282942d89178b4d1f95746940d764112edf39f2280eba6cd64ae2077be39dc8b28717f25
a19b46d3108d61b3a54afd4b6040034c87503205ec83b52cd0d2e224332a8894ba6eac73b176b1413
07da4fce2bffdfb4c0adfa1d7b474f8041bcfca52f1b7838963b55eb2f8446f247cc8605e3b39a738
6173342b5ce8189816e572031d790825f9635bc5eb90ff7deaa1356bfa800ba68d8ed5fe84fb4b19a
33cacbeb9ce10455a19b0e64795dfbdc076409e221d327d79ede5e90d1ec7c4b70e79710f31f668d3
594280ebfcde09815011d5722eff2b9bcc21d753e67f1edae642f957470973c2bc5ae1884bdb0ec25
8ab0cd358cf6c8a7df7fb5d3bd4fd6369bda5ad2af8f0fe014a891232fa86fce37f67406ffdb889d3
7656111d6ed7fbc0f8cf8d497c7a678ea657bc7fc578b2f5dc56e8d24400eaaf945dd6a040de50858
15792539c16354d895aa52345eab5965a439f30cff6bce9bb753bbbe0b317eca4ff6e94ff425753b3
0ebd3e80ed0e75cf1e3f8632b1c646aabba6ea7ee0e23f8bc443e07322c27da24dcd9eab00b062a38
68b0fbef189e02ce1169f477f42ba7c1b7044e5162afb926b6861267c20e4233b8f33641ecb39e9d0
f5e2d286081dfcd12e32a259d24e9db3627520cdc291a9f0ec18d216c4b37a99fa901d64b315893ee

325c8984e07ccb6b6424000856c805d98f1936f8ca86e979918580d96cc1f0d286276c505f6189ac9
925afb7a62b6ec46cc52f8fc34b44bac251d343d00838011f4daff9645bc9759e872f07b77bfdcb79
9389ca12ab9284b955d31db6f06365f48ae860972b29fb2c9d1762c0739dc62f0886ba0764370da16
3b4b4ee875f51501c124938555b1e57f179b7fa422d42a00259bcddb2173bf63fffc2283bec54c17d
030d860f8b2d82c2890fb60bb06a096bc6a0012bfca132620e064727a99876af427cdcd130386abe6
f9973c4c28d2b284bf083bdb19eed3a9e0b38d6144e503ff978cba087d63d8e80eea8b0c2c12960d0
87caa7ccb4d5e2ab5f2998eb13f42ec12f48db6c7aa9fa212e7ec7ff58dee656ebeb2d91f639ed397
028892c993087c51418d021c4090d3d0b0895d3e672d9279df9555236bb5dc7dd4cbb2c410de6f75c
bafa437ac2938cb9ed13b52dd1b92b06ad3588a63e2265dbf7c6541796a8d69b912cd7268b3fbd79f
01d3f5ba4b659c205652a553b8edb04d249ef2cbb39b5e74404729d8e8f0f23f2edbb247f0a74cbad
7580000704bd9f484d0322e05ad977db933b35c71fe252f5ad26d5b31b69c5f251d8621b3fa1d4216
2b7ae3eb5a2405a7b22ae4d4c252e163731b61d027cec273683b5918b6530d264261a75ac92c5b2c8
8ba4565fd15a8d7a95ec698c0ad6b1d3dda674e20a09c810bd888c5c31be5178c453f4614dfbaedf0
7a2ad70c55b5b34f821458a60f5c05018f8a7ccbf7069ffdaec5eca03db5ed37b6656430b07210e37
587b7073a2daee9b5cee325e8a79329867e39cbbc3da01c1fc6061476dc646e3b960cee1238475be3
8085fbe02a886d9cd5f9e64afc4d480f9e743481039e3aa046d2c03072fba6e4df257b965444ef148
938f1965d3e243fb5656e9d33a9573bb86cb0819858d7c8613d9312c939a22b8e7cbfd7719d0fc215
5d1ab55779504ddeef5e03e690eae0fcfc24e1e7ed0f3d76f37744c252743d353428438293570d58f
5614b66e52b31b110428e956b338cfdd8a97a8081974ff58b22cd132143a191039f03212cd6ed075a
0035901c7973b8b2da39175a9b18a5b2e9fabee044331044dfcb6c34e8307aa09d7cccaebaea62a4c
89a9368316dd0771f42e752a572653c9d569fcddce49f2733a90991ff6b83a73397300fbac240ae61
230a54b0f0a48a243f249cf60e726f5599c8487e7b53aac9aa77353a3a2cd016c04f153562ed340ad
a4eb33109374952fd05e772d48109f01ad0f3ad6138ff1818cad06a9cfa261672d735dbd23757cc69
021b153241988188O7b244cdf49d1d3ec99b6d6dc1944d66ed318041174522c7e9b613c44044ad360
6eb47176e54484bacad779e1db8108779a5fc791d169ee1cfb52800c5c583a8da0cb710abc9de6e7b
eb398cd0191f948fd86493f36fb9d8e672284408721a87a74caf7f5fc9bdbe1efdc0117c781fefd56
3a48d9b667ebd462e78a2359c426cd4e0a0637863f14e70d58bd5867881934ebe0386c2693bdcac38
6bf6caf0bc6919ff4910cfb6230987fe8205ae6b7b03d2eb1d6789b2027cb6fa817aa68966500afd0
e8e9588e720d7d0977a7a04b18e272da7741d85b8ff133ff66890c8b64f3f493573bdfb008a24c3cb
8c0718f616f4d34cd5f19d447db5e23434dc246324171785907d52e1814efd65f4f5be845aeaea695
907075d451b81af0d0b82c3b591100fc42f59dd2eced3ee4c2b2b69f71eaf6271ee7d0eb5cb00715f
90f340a9a947efcf59f42206d6bce4031b75c2143712aabfabffc422cc82f60d25737cb0fe924ed81
dad77fb57529034226699f2a34203b705e36ff581d140850e6ac2b3230ab00b7f31eab100d6265080
654af242f34c02b30030aebdf73250ec18163e4acbe1e9c54f2be0bfc25b65262fc6c6126aca73731
b65fe918125640d5cdcbb6df238fed79f14e1ce7c4f4ed4f024fef4655759e1150dfbe87cc2206ed6
cce9b84012516236298O8f79a3d1bc2e5185fc89624f43cc258043a33a9e2df9051ab67c46393822c
01e822c9806ee71d0d1abd3f0f99652d1640489917ecaa8ee70308395cdf9edda7f3bbe71d7f24fa2
078c385a7b674da7a79bb982e28b658e3cd5394c9fe70108adf1025ec2befa962326acf493b753783
129781d3729b57099729c505d02478b91def49d949da8c9a649e203b4ffda40eaeb1b93ef8b81ceb1
1b77649fb1c13d818cdba788d7fd5e7389ba1a4eda5db5e3b0b43228cb4f0709df170ea113c4987c5
4823c0f2465d54f16822d7e6a7aa45714b17322f36f0d8c6e197f6107d9a6db7d2bef35229333ac5a
5f01228a118ad54c7550c8e9dfa1810a2d214c336007b79d501f9c6fc62d6d9c87fbc7fda22e0c94c
3dd0078d95362efeca025b6667fa23bb581764589424c381dd78432bba88968c3ff6f8f4408694feb
72d4deef48749a5ec8fdb4ff007482cba0a339a766b3beb17602c416d5f87395f780935775f9b52de
5c9cac5a76b1494f88757bcc297f5ef370e1ad2f2de0825d6f6e10a8520fd2689ee909644949c76c9
a4253a895fb748f60ec39e51243899e0957388b37148c0b2ec20a950a07d7b1773d3a0169ea64b7e3
cd49c60b77efc7320694cf64306dc099f5ea483323ec0dc1a12cba93a587a535b800f5607e4c53fdd
eb2c3ee20d3626d02b7d1643565d9cbbc21b57f0ccddce0cf31a5bb87914ef721248fc55d4034735e
68700d978fe8e94f789f0bb81e2b02bb441f4f50aeeef72bd504230f5d7dca932b16adee34898b903
3ee8222c4dce4b88d97dc350d2e7433d46736973350201e54f36c47fd01aa2993f5df36bbef556abb
4a2a3fa70e01729b06704697ba0de9daf66953be1f8aaec287e36a7eff704a57a44fbc47dcd21fadc
6a01a92cf3f85e80c78753b4d94bb964174e67c43317318506e7a299b3570da08b94a80f027aa2982
d4e23938b4212091d286b10bc6f8f9e4ae9e2d743f6a17220624e91254409fc44e372f2476af8bb85
109ccf9265aaad431979c18b0c09807fcb9b950137638b059fd5ff9d7c4ff1401f82998460a965d08
d0caf8910b4b310c3f32adefe6feb2e8f546d52b29344fe1d96b0aa31f34a1ad2f8d9c8f2d1c663c8
0edff2a0f0290ab0a264356913047262O3f66481250c7e63aa26b93ed68c543d4bdf8995f0062b573
8867ee444d0a88d7470e520bdb7ab6254c2168081ee60b5c4b51e3a9b5768fb053cb11cbbd957067b
07fb5c821c4a03b94b60fe5067d6fdce799896df69e180549bb1a6d2f575fd9ffa77614e530173b68
a17dffe25c0f2e3ff87f1bf0d04d7ea8ca22d7206c9f1dab6da341e33145ac6c452d474b82f9ed8df
6f851cf117eca9c76a05836d92579f6e9006637f7013882ad2c9076d2caea98a4fbd5c78dc7693ff5

```
6af40eb8349266b7e5001a9de7eca12b28c5ab1f92bbeab25c6215768b8136cf01ff5acab00ae8b0a
3b494ed075aa8a7d38df19f2fdd95926ac62d11b0a34ee0c2c8e488a7a0879e22ffb3e936811365dd
f45207b718f4c49c73e4e997396c20d58ee38b1012b7dbe8da2e181d397bcb1cdfc004557836ed8fb
884c3f0e8fd6aaef6f4d090b7e5e61b7ed3f73be3097437672b92083431dc760e2176e5f7e8fe77c7
dd2ee1df0816a8ccc57d2b9d343117c3da956539d056198232b5423939a3b72386d7abf8eb4b5836f
72e6e1b17e9c1b0c575ce2ee1562646f22a40bb34f93ca8127d3e100857b7376c2ceda57700f78805
e1395abde70d9ee2416c452ead7586c4b0dd3785d0f064f61b529a47891e44e89c4bce7e271f461ea
7806cbd926db3d74539ac8b

Mensagem: b'Mensagem em Dilithium'

Assinatura: 13c68849e478c738c28e436e5f1bb37c40d3f66aa876b9c6561dccdfcee1c7960008b
2dcb49372bcf2a11cf3d23bea1c550b426572d8e9cc58bdd3fbacce14de026c9feca04319bd0557e9
6f9f7b2434a86bbee376972c2d40a19e9926e4207ba6eb7b5de421d227a7afcb1a1c09d32ad045fc3
dcc9d59209d8bf708219553b54747c26d84e02cd7f3a7dba0ca4333f2a7238dbb7d0b2d975b403762
c0a78d87b4186f0db73522e70855fddb6eae67b44c85b9345fce4a95893fae094f2a9126a60c94754
3d63ddd6899d6dd498842b9a8a1a2eef63951e9f3a7b757b7445721687b1d430f7ecf92546de1f995
bdfbd32ffbf6720ea53dbe729f88d9f4b8270ed7637424fc8147704ca86b0cf7806f4ff9f5f0031f5
b23e0f90376fc8493e6a9e65d1df29ea8cf5d376d7cbbaf16ac78818745250893ea7732e69c7e3922
f8a8afe2b735a6a69ade499ac535ca53f6f7a89e5c68ae08706937b7e3965caf8887b346ef6f9161c
f72ddc8e234b34e9033fdc9c285b2559f66c53269e9d6c97b397bc64a59163f0f55f3cf79f63b18bd
0ba0f3c1dd0439c0888b05cdfda409fb2f9139d3424e0116c1931f54b397a0d17532bcbf111402139
6276f62d7736e62c831b6274fbc266d63bb984bcb7b013a7bd231e8b6d834ee964ae16d427cf84b9f
3cee85cb5ebc583630ac4400ed27b2b7bcc341dd2910ddefe2f7e8c3943524af0a0c79dc787101ba7
b9c2ed37f744990b45bd3ff58b35c815b47ad3b628dd64caeef34448ff8143144de59f7e798fb27b1
0c64234a793b780bbbfe4704a0e54047d7f714c16dc0073d9fc91326c1d60e5a12906bd78c893e947
ea02b3e4b7e9b47e1731bdd96fd9011185e36912630259ad4195fa8fd1245c19b85c37e65490c8258
98a8060f141076c436a99e37413b1aa62f92f3871d05c2aeed9a0fbd57daf1ad84b968df49a1ee6aa
f0eeef058f724fcb25cd193a054293f8fae12632cbbdeaaef42ae3cdc2575494136f42e3ac6a11c77
f42a396c034d5105276a8f1290828704c4e3591cc48dc170edcd3c88a94f46347d15a9b891f30df06
eec94c732bb930593b39dd584123f36db8dbf735087cdf508104534686e1cf19c2cf6dc3088fd0111
89cef9d10dfc5c9af2f3190f4ebae75fbb8191339b33903a058f429049dd262fd4476829383315b0a
2d5cb83107361de97d31a601f1059024e4d3645a5a5bd4f1e38b62dff75cdd00c466803a1fc04a154
98260b55ed4b736200e813c4f900e61155d5fa47af94d4ea11b58f065da1ea4ab0eff1902eef5514a
ac12e78e7a36eacc1b84b1d3a09b8c2f31ef6a2cfd647e8d6c7b3fa49067fe5dcbaa34a76e9ddd79e
dad7065abe69d4856e3341092c2c158b8502864c533c598f3548d25119abf1e2f53f9c9aec1afdfad
5e6e7032e84df054fe131a5b3a847262882f7ce2b3c72dd85f3be8882fae8d186a598eeb553922aae
c076d969bd7530a5e3d4b0e35bcf493870a27224c54bc56c5a767d1482bd6e767bd6446951b905e69
db45de9a6b305244224c01c71b1effe53d28aa44736815bf2efddc5e0baf28b71e2c5b1d7f98d572f
fb0fda9935e67c9f899d5a849ca75fc330fdd43201ec9b4ede2aa78cd9a01e31786779add952ad9b1
bf8560f6ea9a39e4e67223e923d693426f2d22f2bf9d15f5e3c51da1491f9eb3b6f7becd2cc3484b0
5cec131008e605f9051a948f8c633a2b8963275cb3e43c04554a9b6207c6918d101625f8c8f69ba75
5c5c9b4642acb92874fc3d0effc2c33315cb237a361c889c36a62663b900c3ca38b30d5d7386c7c1e
0e4509d806a301bd2f3b503300a18b32322c0cf78f35bd05019aa01c4b33c468f433b4a15b28b2e55
08f73a85b8781e3baa5e8d1ccfc92ece6f63b0b3e24427e807e37defe588ce442d92f6d6e958b6089
d330c570da9f795222bc9d822a52d7228862bcec1a8d9261e9387956a58f91ac8f85b51fd8792d439
7fa6cb40a5ba0f493f41b4367f6dedf4b4fceae22f49c4fa313f3abe5bd2dc7eb99221d4c68681250
7e26062de5d66dadb7035695901a1a86f2da14fc0cf53c78a1057f077161b8368f9ea52ee9d639d0d
5c0b4623017a1512980c56a0332f5bdc2be3f32df77de9047d75b523c67f2c26a72f2827e5597ebb7
b0942f08db03848815f4cfe85c2ee8d4b8cfe9659f7f447ca46accfdb9728926597378c22c011ab3c
20b2a7d773633411b4f54babb0f10f344c2a05f57376884058ed6e7013937bf2d2e04b828eb6f4625
312b1bb673e4fa102dfbe7efbe70719570b15a6e4da146d62436d50c74f2004d2e1033ebc03b39721
e604318d2cc6198fe899b58db41d5f3819f719b91ac8d073d68ce7ccc74409019c76fa4ecf513bfb9
4350c2832bd8aee940a030c5d539b59afd701bbd8f255c7396a7a7230f3c823eb9e468d122abe3f6d
c20cfee3d554b37a3374a9c5775d44fc0bca092888b4b25c0f304a7d5afbe9219c3929df115954e90
5b7b72fc66a9a46cf15f7c04a1d6335d88261b5e15c46442422809b6a879fd2703e9a1f1f6adee3b6
d4f973806c917be83fba106c824d02adf687ebe09ad08e1eeaaa2892bfa0e3da1e94dd04f32f1fd76
bbab3af92b0ca3376a3a7de525dbe6137c61ab53a1e581c22c98f342d65d3400c7f309745329f6662
1ae1bc6fb298a5b48973d1f4af78c3e94913c274c0630733574de4b5c3389222b1832488f48f55b28
e2ae8d56b400a4edcbc50bf56ce7d6d726a5315d1013c04909a21c2aed5a4cf13cb58a903d6e94f46
```

1bcb36f3edf604d0d2d2190cfdc2dc808354e84e99338372ebd9cca3e8098c6b6507b95f5aab6ef0a
4acd1d9c61d02f17d6b95fc1b0e74ed0666c1634760f578d0a9571ff60f74c9b23d5f21f483014f18
3625983558e3253affba71e405d1784f9959ed049cc2efdc08eb04395b858dfda840f5760576b3693
39ca82203a4ebff430546ddc75575375177aa73feba7bc026dd30fd9fee9a2156feebb082e87dd529
c18cc852b6d7f2600951e7fa51fe38fdd98db93c7d7bf39d788a9ff2bb7ce5d3e061226da8bf79b2c
cadc5d1c47228d7f5babf4c275374ae33f88f0247b3ea78ae4836459c54fbad17fed3171af3b24119
59809e6082658ad435ce88d34b2868424dc34cb0c03a543a4353ab3ae4fedd87a304b76e81b4df729
a69ac00af2a8e0dd136c47662371aeb843e1acfde17136a96ee8cde6b5a9ea68aa0d3b54fae795fab
9ca77eac1787543d969a6bc66194f06e319aef85b352cab7723952ce42a796495679d9dcd7ced45b1
5820037cdc8a2cd7b6651aab2ec3a1cb6c1f5b1721f66bcd5e2cb69d99b6a1e7baa1e5d5a452c010a
bd5f66c0991b9e06ccd13bca3c083561a9e82459bc437aabbac11d129ed60cc843923a54ea9b05a11
3bcfb65864246a918e04f82d88294333c424dc051f4c84ee8d1459e3c8dd1534cd4ce4dd1b1274df2
f56605545875d45b476108c4bef04122ac770ca418171bc39c12a26c926cba153af75427f618a3d9d
978405eb39e8a14bd43dbda90d7bf8ae8e3dfa89417bf369dde548b8f256864ea1d71d6dfdcad90e9
2d65a39a65f19b3d04c880ce0103d0f5f12f3b5800d7c16514848c95112c2f15bed94f8f1e0118b65
58704b42008ce70873e81d7b6653283f56dac5eaac09ac893dc1a5e9f9afc35f7f2fae8fc091b8ef0
811be45c241ae05934bd08205b2ba675cf4e067437e0d01f3023d30d619abe1ff3da85aca709eb169
eedb8577416a8cf957f11e1ab082cf00cc3c5fb4ee8c4ae0752643b6a6419170160bd26621c5046e4
17f06e95b40336e00ac2ff56cffe7769e575aa1721d164571ef99dd34ea36fc171eaa7a9959ace69e
8468c8fd6b66d880e2d70c73bd7cc7ed4638bbccfb8d1517f01fb1f0f65efee1d40edffe23434498d
81fee134b5be107f8b0eae9064303fd8bb68162f42f952436957a09e0fb73462c09ddd9a41c1fec2f
7cc0b98ea1cb2c0b419304a3a2148a187e83d1a75a5d008888ac6b7ab8bfa6ccf00fc22c55dc2b9f5
f43f10039a20a7dc27bd1fded438303c7c92c97eb33dc332af3b4e7ee3e73008974eb69b8a0c34841
97e73ce9edb80f7f1dc9c1210b7296de58460ef9c466dc7d4e8d9b58d6665c78bb4959cc87d5178ed
aee128325dfce296d2f4e9f2e5ba2699a3f7b41f7052ae6f79114e6a0208b3b39e069fa37901f80bc
91e4cca7987ba6034b7625214aff48a81eee7c8c886af2528d19004256330034ba6b696dcc857daf5
bff1aa3c2470d65bab5dbb596f0b7a6085079a1d73b6e160c84676f0b701f48716fea210e84f8a165
babfa5fc6e1ee149a4ef17b4a6b6ca123db0df9c2f2fa8a0d0a643a7dd1133a707417e691afb0e1eb
db83e669734130b02c80da60eac818449577dbf5fb0bc34b6bd9ebde9a7ca0dfd5c9f16f6844ba899
6166fd57f1d42041ebdc8dbf50a7c3ee20de064a2c254374048b09a367e948191abe60bce804b5482
e104b0932c00d4754d2bc5504e69a5bc2ea55f2cdd4a8a8b3bffa1fab7ed593ae7788f0ef2caaed2c
09c036972cac57597b73c7fc494d210cdc68c67865e68ca9124d342a48710a227915f31ff4329eb4b
1937b80a754f8f1d68eebee54fb3cfe0fe95028d7bcb80a26b0b51fe4bb371f0c68efc033c445768d
62ca91a68f2a083a30573b243ae978dc449a06d56a4ab48e7899531f3c4bfc35dede94e0db7c0b7e8
058fa14eb1219a561c5e4f37ca5a361bcf5a4a327fb8f173df98895d394bf963a741c0fe9b5c8771a
0e73e1d7428c83bbfc2c1239b5b4dcc0a146ec5d21f49cd2e425db2b56b782fcf081eb0218ca02843
02761f3e770113b9f8aef304bda22cb0c2efb73af84da9a6393917c0012a79765f1464302a41c43c9
556a68abc17f739349be7ece8efb104607dd0de578d09fc0aff942da3da0e7a44061659edb0ecf638
16256f97134053c0e5cd2339dcb9a886a6ecb037b26b1b3cd0ccab4f68f7031ce74ea0a072ffb95d4
5d90112077472e6ae757e824639cf2fed2b33cb715882e13f6ab174d0a7f717b2667956b174af9e8b
98ee7778e49fbab8e9b232d68e839164946f07530b9ff82224f19f31b8fffe54f6b919600bcf1d880
84656c451896b26e302b0def5b9a3f7a1ff64a995acc24de86429d3eaae520da97641c778376217b1
79f9cbacc7246657c54e16649ed0bd74dd69187e8216106c675febe65c37d6c9e74dee2582d67fd02
ee82d7c82eae580cfad923a8990cffcafb6366b7757001084442b88e4589906564b5020eba00a2b57
b84ba33cfdb78fed3a3e568673eaf878c0bf493f2aae8ff5c9ace7471cc8e571bb9e485d299867a17
828c45cb7b2d7ea934b0f5e8ca54e320ed684508dbb6d10ec26d05d6c227ae8999d0e567ea702bec8
bf858cdb4b7cd90ce4c8834c594a993f443459554275c0bf6c3e53f525e6a45698db8abcc5fba7848
a132cbf9f78a704d0c122e551581f8f9958feafe4ac0994fbecca1a93607da728638ad8af09d046be
33f62ce3cfa85a1880b0373eb2f777e328687f0d034df79e34fa1ec5a4d222f4566abc690b954702e
38d412226edaa2ef7b3831d7616b95e336b5a8e6e82f790cb5ba648835114088e7bda826ca6e92986
1f3118d4d780ceaddf8645ee7cf6ab69b7df61a99482d5d6cd4385781bec5ebbb6aa8e3069e269aad
c67901878354e8862fece666bc318e1e245f58d1d4c53d2af2b2ab27437806a32fc4ee1562ce89f84
cdbc27d858d9dc6e2ec81d64b900e176010d51e4b0df38118a392f6cfe3fa59657a07b7509f538ce4
45ef18dabd443d27bc7d51d7a3bca44436b0333fd481c8b87e5811dc27e0394e2592fe218cc956d85
598fc471523d78b09400b65128943688623e6fd39d73c37cd122f4f68295a7287b85addfb726c053c
5645b174f7a4c3631adb7f172767e1ac142ed89c37c316d1f8f3ce2648396aa4eea418ca8fd55929f
908039b2c6c33e9ea0e32edb131a6216c5ee130d78bbeef624711382bda37e0e9891d4aae9351b904
16c032d82eeefd9c54c3675a640e40f7edd01bdd50e2c1fd248c1ba27b77795c317d823011af86fb4
5d75c83223c1054334e062c2ceefd346d6a6c6b83d572f21cc484a457aa9504a2b83f455ec1184f98
21d29ef5233da30d9e84cc96a6903fca5a8321f94193400fa0c9c57ca26d5540a8883cb83201325bc

```
e6da8f1ad862c6c4c4e5389c29fe111a93c2e107e8f810d64311e735e2d7e50dd6359ae816953901d
4eb1041860114169c082df6c8805a105391834f3d863f6ac35778abeaa505fdba6ac870b71095e2c2
4ad237de46f37b009f6f6e4e2ff8b894c3db5015df08e3a4b7adbdee2e301494e6a6b6ea1d2081b31
5068a9d6e1f60812217391cc166998a7d5212829649ebabbd6e001294a649697a1b0d2d8e2030c70b
7c2d0000000000000000000000000000000070f181e232c373d
Verificação? True
```