g13-tp4-ex1

May 28, 2024

1 Estruturas criptograficas: TP4 problema 1

1.1 Dilithium

Este é um algoritmo de assinatura digital pós-quântico que nos permite perceber se aconteceu uma alteração não autorizada, ou seja, o remetente poderá utilizar a assinatura digital para provar, que uma determinada informação não foi modificada e que a mesma veio de um determinado emissor.

1.1.1 KeyGen

A função keygen é responsável por gerar uma chave pública e uma chave privada (Byte strings), para que possam ser utilizadas pelo emissor e o recetor, para assinar um determinado conteúdo, e verificá-lo respetivamente.

1.1.2 Sign

A função sign é capaz de receber uma chave privada sk, e uma mensagem M (byte string), e gerar uma assinatura "sigma". Esta assinatura terá toda a informação necessária para que a função verify possa verificar a validade da mensagem relativamente à sua integridade.

1.1.3 Verify

A função verify recebe uma chave pública pk, a mensagem que queremos verificar M, e a assinatura sigma, que terá nela toda a informação necessária para extrair os parâmetros para a verificação da mensagem. Após a extração de todos os parâmetros necessários, e consequente verificação, a função retorna um valor booleano True caso a mensagem não tenha sido alterada, e caso contrário retorna False.

[324]:
```python
from hashlib import shake_256, shake_128
import os
from functools import reduce
```

[325]:
```python
class DLTHM:

    def __init__(self, security_strength = 2):
        # ML-DSA-44
        if security_strength == 2:
            self.q = 8380417
            self.d = 13
            self.tau = 39
```

```
            self.lam = 128
            self.gama1 = 2^17
            self.gama2 = (self.q-1)/88
            self.k = 4
            self.l = 4
            self.eta = 2
            self.beta = 78
            self.omega = 80

        # ML-DSA-65
        elif security_strength == 3:
            self.q = 8380417
            self.d = 13
            self.tau = 49
            self.lam = 192
            self.gama1 = 2^19
            self.gama2 = (self.q-1)/32
            self.k = 6
            self.l = 5
            self.eta = 4
            self.beta = 196
            self.omega = 55

        # ML-DSA-87
        elif security_strength == 5:
            self.q = 8380417
            self.d = 13
            self.tau = 60
            self.lam = 256
            self.gama1 = 2^19
            self.gama2 = (self.q-1)/32
            self.k = 8
            self.l = 7
            self.eta = 2
            self.beta = 120
            self.omega = 75

        self.n = 256

        Zq = IntegerModRing(self.q)
        self.Tq = Zq^256

        R.<X> = PolynomialRing(Zq)

        self.Rq = R.quotient(X^256 + 1)

    # Função auxiliar para transformar bytes em bits
```

```
    def BytesToBits(self, B):
        b = [0] * len(B) * 8
        B = self.BytesToByteArray(B)
        for i in range(len(B)):
            for j in range(0,8):
                b[8*i+j] = mod(B[i], 2)
                B[i] = B[i] // 2

        return b

    # Função auxiliar para transformar bits em bytes
    def BitsToBytes(self, b):
        l = len(b) // 8
        B = [0] * l
        for i in range(0,8*l):
            B[i // 8] += ZZ(b[i]) * 2^(mod(i,8))
        return bytes(B)

    # Função auxiliar para transformar bytes em bytearray
    def ByteArrayToBytes(self, B):
        return bytes(B)

    # Função auxiliar para transformar bytearray em bytes
    def BytesToByteArray(self, Bytes):
        return list(Bytes)

    # Função shake_256
    def H(self, bytes, length):
        return shake_256(bytes).digest(length//8)

    # Função shake_128
    def H128(self, bytes, length):
        return shake_128(bytes).digest(1024)

    # Função auxiliar para transformar inteiros bits
    def IntegerToBits(self, x, alpha):
        y = []
        for i in range(alpha):
            y.append(ZZ(x) % 2)
            x = ZZ(x) // 2
        return y

    def CoefFromThreeBytes(self, b0, b1, b2):
        if b2 > 127:
            b2 -= 128
        z = 2^16 * b2 + 2^8 * b1 + b0
        if z < self.q:
```

```
            return z
        else:
            return None

    def CoefFromHalfByte(self, b, eta):
        if eta == 2 and b < 15:
            return 2 - (b % 5)
        elif eta == 4 and b < 9:
            return 4 - b
        else:
            return None

    def RejNTTPoly(self, rho):
        a = [None]*256
        j = 0
        c = 0
        while j < 256:
            h = self.H128(self.BitsToBytes(rho), 1024)
            a[j] = self.CoefFromThreeBytes(h[c], h[c+1], h[c+2])
            c += 3
            if a[j] is not None:
                j += 1

        return a

    def RejBoundedPoly(self, rho):
        a = [None]*256
        j = 0
        c = 0
        while j < 256:
            z = self.H(self.BitsToBytes(rho), 2048)[c]
            z0 = self.CoefFromHalfByte(z % 16, self.eta)
            z1 = self.CoefFromHalfByte(z // 16, self.eta)
            if z0 is not None:
                a[j] = z0
                j += 1
            if z1 is not None and j < 256:
                a[j] = z1
                j += 1
            c += 1
        return a

    # Função NTT
    def NTT(self, f):
        f_ = list(f)

        k = 1
```

```
        len = 128
        while len >= 2:
            start = 0
            while start < 256:
                zeta = mod(17^(self.BitReverse(k)), self.q)
                k += 1
                for j in range(start, start + len):
                    t = mod(ZZ(zeta) * ZZ(f_[j + len]), self.q)
                    f_[j + len] = mod(ZZ(f_[j]) - ZZ(t), self.q)
                    f_[j] = mod(ZZ(f_[j]) + ZZ(t), self.q)



                start = start + 2 * len
            len = len // 2
        # f = 8347681
        # for j in range(256):
        #     f_[j] = (f * f_[j]) % self.q

        return f_

    # Função NTT Inversa
    def NTTInverse(self, f_):
        f = list(f_)

        k = 127
        len = 2
        while len <= 128:
            start = 0
            while start < 256:
                zeta = mod(17^(self.BitReverse(k)), self.q)
                k -= 1
                for j in range(start, start + len):
                    t = f[j]
                    f[j] = mod(ZZ(t) + ZZ(f[j + len]), self.q)
                    f[j + len] = mod(ZZ(zeta) * (ZZ(f[j + len]) - ZZ(t)), self.
↪q)

                start = start + 2 * len
            len = len * 2

        return f

    # Função auxiliar para inverter bits de um número com 7 bits
    def BitReverse(self, i):
        return int('{:07b}'.format(i)[::-1], 2)
```

```
    def ExpandA(self, rho):
        A = [[None]*self.l for _ in range(self.k)]
        for r in range(self.k):
            for s in range(self.l):
                A[r][s] = self.RejNTTPoly(self.BytesToBits(rho) + self.
↪IntegerToBits(s, 8) + self.IntegerToBits(r, 8))

        return A

    def ExpandS(self, rho):
        s1 = [None]*self.l
        s2 = [None]*self.k
        for r in range(self.l):
            s1[r] = self.RejBoundedPoly(self.BytesToBits(rho) + self.
↪IntegerToBits(r, 16))
        for r in range(self.k):
            s2[r] = self.RejBoundedPoly(self.BytesToBits(rho) + self.
↪IntegerToBits(r + self.l, 16))
        return s1, s2

    def Power2Round(self, r):
        r_plus = mod(r, self.q)

        r0 = self.mod_plus_minus(r_plus, (2**self.d))

        r1 = (ZZ(r_plus) - ZZ(r0)) // (2**self.d)

        return r1, r0

    # Multiplicação de matrizes
    def MatrixMultiplication(self, A, u):
        aux = A.copy()
        res = [0] * self.n

        for i in range(self.k):
            aux[i] = self.MultiplyNTTs(A[i], u[i])

        for i in range(self.k):
            res = self.ArrayAddition(res, aux[i])

        return res

    # Adição de matrizes
    def MatrixAddition(self, A, B):
        res = []
        for i in range(self.k):
            res.append(self.ArrayAddition(A[i], B[i]))
```

```
        return res

    # Adição de vetores
    def ArrayAddition(self, A, B):
        res = [0] * self.n
        for i in range(self.n):
            res[i] = ZZ(A[i]) + ZZ(B[i])

        return res

    # Subtração de vetores
    def ArraySubtraction(self, A, B):
        res = [0] * self.n
        for i in range(self.n):
            res[i] = A[i] - B[i]

        return res

    # Multiplicação de polinómios NTT
    def MultiplyNTTs(self, f, g):
        h = [0] * self.n
        for i in range(128):
            # print(f[2*i])
            # print([2*i + 1])
            # print(g[2*i])
            # print(g[2*i + 1])
            h[2*i], h[2*i + 1] = self.BaseCaseMultiply(f[2*i], f[2*i + 1],
↪g[2*i], g[2*i + 1], 17^(2* self.BitReverse(i) + 1))
        return h


    def BaseCaseMultiply(self, a0, a1, b0, b1, y):
        c0 = mod((a0 * b0) + (a1 * b1 * y), self.q)
        c1 = mod((a0 * b1) + (a1 * b0), self.q)
        return c0, c1

    def round(self, x):
        return int(x + 0.5)

    def bitlen(self, a):
        return len(bin(a)) - 2

    def mod_plus_minus(self, x, y):
        result = (ZZ(x + y // 2) % y) - (y // 2)
        return result
```

```python
    def SimpleBitPack(self, w, b):
        z = []
        for i in range(256):
            z += self.IntegerToBits(w[i], self.bitlen(b))

        return self.BitsToBytes(z)

    def SimpleBitUnpack(self, v, b):
        c = self.bitlen(b)
        z = v
        w = [0] * 256
        for i in range(256):
            w[i] = self.BitsToInteger(z[i*c:(i+1)*c])

        return w

    def BitsToInteger(self, y):
        x = 0
        for i in range(len(y)):
            x = 2*x + y[len(y) - i - 1]

        return x

    def BitPack(self, w, a, b):
        z = []
        for i in range(256):
            z += self.IntegerToBits(b - w[i], self.bitlen(a + b))

        return self.BitsToBytes(z)

    def BitUnpack(self, v, a, b):
        c = self.bitlen(a + b)
        z = self.BytesToBits(v)

        w = [0] * 256
        for i in range(256):
            w[i] = b - self.BitsToInteger(z[i*c:(i+1)*c])

        return w

    # Codifica a public key
    def pkEncode(self, rho, t1):

        pk = rho

        for i in range(self.k):
```

```
            pk += self.SimpleBitPack(t1[i], 2 ** (self.bitlen(self.q - 1) -
↪self.d) - 1)

        return pk

    # Descodifica a public key
    def pkDecode(self, pk):
        rho = pk[:32]
        z = pk[32:]
        t1 = []
        for i in range(self.k):
            t1.append(self.SimpleBitUnpack(z[i * 320: (i + 1) * 320], 2**(self.
↪bitlen(self.q - 1)-self.d) - 1))

        return rho, t1

    # Codifica a secret key
    def skEncode(self, rho, K, tr, s1, s2, t0):
        sk = rho

        sk += K

        sk += tr

        for i in range(self.l):
            sk += self.BitPack(s1[i], self.eta, self.eta)

        for i in range(self.k):
            sk += self.BitPack(s2[i], self.eta, self.eta)

        for i in range(self.k):
            sk += self.BitPack(t0[i], 2**(self.d - 1) - 1, 2**(self.d - 1))

        return sk

    # Descodifica a secret key
    def skDecode(self, sk):
        rho = sk[:32]
        K = sk[32:64]
        tr = sk[64:128]

        v1 = 128 +((32 * self.bitlen(2 * self.eta)) * self.l)
        y = sk[128:v1]
        v2 = v1 + ((32 * self.bitlen(2 * self.eta)) * self.k)
        z = sk[v1:v2]
        w = sk[v2:]
```

```
        s1 = [None]*self.l
        for i in range(self.l):
            s1[i] = self.BitUnpack(y[i * 96: (i + 1) * 96], self.eta, self.eta)

        s2 = [None]*self.k
        for i in range(self.k):
            s2[i] = self.BitUnpack(z[i * 96: (i + 1) * 96], self.eta, self.eta)

        t0 = [[0] * self.n for _ in range(self.k)]
        for i in range(self.k):
            t0[i] = self.BitUnpack(w[i * 416: (i + 1) * 416], 2**(self.d - 1) -␣
↪1, 2**(self.d - 1))

        return rho, K, tr, s1, s2, t0

    def ExpandMask(self, rho, mu):
        c = 1 + self.bitlen(self.gama1 - 1)

        s = []

        for r in range(self.l):
            n = self.IntegerToBits(mu + r, 16)
            n_bytes = self.BitsToBytes(n)  # Convert bits to bytes if needed
            v = []

            for i in range(32 * c):
                hash_input = rho + n_bytes
                hash_output = self.H(hash_input, 1024)
                v.append(hash_output[i % len(hash_output)])  # Collect␣
↪necessary hash output bytes

            s_r = self.BitUnpack(v, self.gama1 - 1, self.gama1)
            s.append(s_r)

        return s

    def Decompose(self, r):
        r_plus = mod(r, self.q)
        r0 = mod(r_plus, 2*self.gama2)

        if ZZ(r_plus) - ZZ(r0) == self.q - 1:
            r1 = 0
            r0 = ZZ(r0) - 1
        else:
            r1 = (ZZ(r_plus) - ZZ(r0)) // 2*self.gama2
```

```python
        return (r1, r0)

    def HighBits(self, r):
        (r1, r0) = self.Decompose(r)
        return r1

    def LowBits(self, r):
        (r1, r0) = self.Decompose(r)
        return r0

    # Função para gerar as chaves
    def keygen(self):

        zeta = os.urandom(32)

        temp_bytes = self.H(zeta, 1024)
        # temp_bits = self.BytesToBits(temp_bytes)

        rho, rho_, K = temp_bytes[:32], temp_bytes[32:96], temp_bytes[96:]

        A_hat = self.ExpandA(rho)
        # print(A_hat)

        s1, s2 = self.ExpandS(rho_)

        ntt_s1 = []
        for i in range(self.l):
            ntt_s1.append(self.NTT(s1[i]))

        t = [
                reduce(self.ArrayAddition, [
                self.MultiplyNTTs(A_hat[i][j], ntt_s1[j])
                for j in range(self.l)
            ] + [s2[i]])
            for i in range(self.k)
        ]

        t1 = [[0] * self.n for _ in range(self.k)]
        t0 = [[0] * self.n for _ in range(self.k)]

        for i in range(self.k):
            for j in range(self.n):
                t1[i][j], t0[i][j] = self.Power2Round(t[i][j])

        pk = self.pkEncode(rho, t1)

        tr =  self.H(pk, 512)
```

```
        sk = self.skEncode(rho, K, tr, s1, s2, t0)


        return pk, sk

    def w1Encode(self, w1):
        w1_hat = []

        for i in range(self.k):
            w1_hat += self.BytesToBits(self.SimpleBitPack(w1[i], (self.q - 1) /␣
↪(2 * self.gama2) - 1))

        return w1_hat

    def InfinityNorm(self, w, num):
        for i in range(len(w)):
            for j in range(self.n):
                if abs(ZZ(w[i][j])) >= num:
                    return False
        return True

    def MakeHint(self, z, r):
        r1 = self.HighBits(r)
        v1 = self.HighBits(r + z)

        if r1 != v1:
            return 0
        else:
            return 1

    def SampleInBall(self, rho):
        c = [0] * 256

        k = 8

        for i in range(256 - self.tau, 256):
            while self.H(self.BitsToBytes(rho), 1024)[k] > i:
                k += 1

            j = self.H(self.BitsToBytes(rho), 1024)[k]

            ci = c[j]
            c[j] = (-1) ** self.H(self.BitsToBytes(rho), 1024)[i + self.tau -␣
↪256]
            c[i] = ci
```

```
            k += 1

        return c

    # Codifica a assinatura
    def sigEncode(self, c_, z, h):
        sig = c_

        for i in range(self.l):
            sig += self.BitPack(z[i], self.gama1 - 1, self.gama1)


        sig += bytes(self.HintBitPack(h))
        return sig


    def HintBitPack(self, h):
        y = [0] * (self.omega + self.k)
        index = 0
        for i in range(self.k):
            for j in range(self.n):
                if h[i][j] != 0:
                    y[index] = j
                    index += 1
            y[self.omega + i] = index

        return y

    # Função para assinar
    def sign(self, sk, m):

        rho, K, tr, s1, s2, t0 = self.skDecode(sk)

        s1_hat = [self.NTT(s1[i]) for i in range(self.l)]
        s2_hat = [self.NTT(s2[i]) for i in range(self.k)]
        t0_hat = [self.NTT(t0[i]) for i in range(self.k)]

        A_hat = self.ExpandA(rho)

        mu = self.H(tr + m, 512)

        rnd = os.urandom(32)

        rho_ = self.H(K + rnd + mu, 512)

        k = 0
        (z, h) = (None, None)
```

```
        while (z, h) == (None, None):
            y = self.ExpandMask(rho_, k)
            # print(y)
            # print(len(y))
            y_ntt = [self.NTT(y[i]) for i in range(self.l)]

            w = [self.NTTInverse(reduce(self.ArrayAddition, [
                    self.MultiplyNTTs(A_hat[i][j], y_ntt[j])
                    for j in range(self.l)
                ]))
                for i in range(self.k)
            ]

            w1 = [[0] * self.n for _ in range(self.k)]

            for i in range(self.k):
                for j in range(self.n):
                    w1[i][j] = self.HighBits(w[i][j])

            c_ = self.H(mu + self.BitsToBytes(self.w1Encode(w1)), 2 * self.lam)
            # print("c_", c_)
            c1_ = c_[:32]
            c2_ = c_[32:]

            c = self.SampleInBall(self.BytesToBits(c1_))

            c_hat = self.NTT(c)
            cS1 = [self.NTTInverse(reduce(self.ArrayAddition,
                [self.MultiplyNTTs(c_hat, s1_hat[j])])
                ) for j in range(self.l)
            ]

            cS2 = [self.NTTInverse(reduce(self.ArrayAddition,
                [self.MultiplyNTTs(c_hat, s2_hat[j])])
                ) for j in range(self.k)
            ]

            z = [self.ArrayAddition(y[i], cS1[i]) for i in range(self.k)]
            tt = [self.ArraySubtraction(w[i], cS2[i]) for i in range(self.k)]
            r0 = [[0] * self.n for _ in range(self.k)]
            for i in range(self.k):
                for j in range(self.n):
                    r0[i][j] = self.LowBits(tt[i][j])

            if self.InfinityNorm(z, self.gama1 - self.beta) or self.
↪InfinityNorm(r0, self.gama2 - self.beta):
```

```
            (z, h) = (None, None)
        else:
            cT0 = [self.NTTInverse(reduce(self.ArrayAddition,
                [self.MultiplyNTTs(c_hat, t0_hat[j])])
                ) for j in range(self.k)
            ]

            sub = [self.ArrayAddition(w[i], cS2[i]) for i in range(self.k)]
            su = [self.ArrayAddition(sub[i], cT0[i]) for i in range(self.k)]

            h = [[0] * len(cT0[0]) for _ in range(len(cT0))]
            for i in range(len(cT0)):
                for i in range(len(cT0)):
                    h[i][j] = self.MakeHint(-cT0[i][j], su[i][j])

            count = 0
            for i in range(len(h)):
                if h[i] == 1:
                    count += 1

            if self.InfinityNorm(cT0, self.gama2) or count > self.omega:
                (z, h) = (None, None)

        k += self.l

    for i in range(self.l):
        for j in range(self.n):
            z[i][j] = self.mod_plus_minus(z[i][j], self.q)

    sig = self.sigEncode(c_, z, h)


    return sig


def Verify(self, pk, M, sig):

    rho, t1 = self.pkDecode(pk)
```

1.1.4 Geração de chaves

[326]:
```python
dilithium = DLTHM(2)
print("KEYGEN")
pk, sk = dilithium.keygen()

print("Public Key: ", pk)
print("Secret Key: ", sk)
```


```
KEYGEN
Public Key:  b'\x15\x861\x84\xfcs\x83\xe5\x8f\xd7!/\xb3LE\xc5\xa2l%\xb6\xeb\xe9\
xe7A\x8ar\x90\x12\x807\xea\xec8\xc5T\xce}v;z\xe2x#\xe4\x85\xd9E\xb9\xff\r`\xfa\x
b2#\xe0\x805\x06\xa6\x89X\xf4)\x18=\x87\xfb3\x84.f;\xeb 0e\xe6\x83<\xf6\xe1\xf2\
\\x0c\xabXlB\x0c\xcf\x87\xa5\xe8\xcdR\xed\xaa\xe8^o\x0b\x85+\x9fV)x\xb8\x1fY\xda
(Y)t\xd8\x80\xbe\xfd\xd3\xefL.\xabxT\xcb+\xd8e2\x83>\x0c\x99B \xdfi\'^B\xe3\x9e\
xcb\xcb,6(\xb6\xcb6\xe3\xdd]}+\x86\x14\x9f\x0521\x8c\xcf(\xaep\xee\xce`\x86\x00I
\xc7\x87\xc9\xd5`\xbe\x881\xf7\x95w\xbe\xc7Kk@\xec\x0c\x05\xc0\x8c\xe0\x8f\xf6\x
d6>\xff+S#\x03\xaeT\x89\xcb\x9d\xe9\x02B\x80s\xd3${a\xbc\xfaR\xeax\xab\xbf]\x8f\
x1f\xd0\xc2\x15\xc2lLdE\n\xea\n\x0b\x0f\xa1cBNH\xa7\x08^\xb2\xad\xd0\xb10\xce\x9
7\xf3\x9a\xdc)\x14C\xffb\xa8\x9c\xbfi\x9f\xdb}\xf6\x9b\x8e\xf7\xc4\xe0\x08\x15J\
xb2Y>op\x05\x84Cr]\xbf_l9\xcf\xad\x9d\x8fF\xd0\x93`g\x97\xecW\xbc\xd6z\x1eZ\xdbw
w\xa3g\xb9s\xed\xe2s\xdd\xb6&AaZ\xe8\x12J\x99#\xd3\x18\xc1\xfb\xd2\x9au\xc3KG\xc
eN\x1f\xee]\xf8\xef\xdb\x1a\xa9/\x1d(\xa7\xde\xa9\rW_q\xb3\x02\x08\xfb\x16 \x91\
xea\x10;\xddF\xd0\x9c\x8b\x8a\xc9]\xd9GF*\xcf\xc1Q\xad\x0eT\xa5\xd0A]\xf9\x9dAFE
\xdc\x1e\xec&\x0e\xe3\xdf\x91\xb7;P\x8b\x82B\xc6\xf6\x85\xc7h\xdd\xab\xbfv\x94\x
91#\xcaj\x11\x7f\xdeXF\x05{\x1e\x8d\xca\xdcaS<\xa2k\xdd\x0e\x86\x7f\xd6\xd7,UF\x
b1\x82(\x8b\xc6\x05JYq\x7f(\x98\x88\xad\x14j\xeb\x95-(y\x90Y\xe1\x8e\x03\x93j(\x
de\x91\x85\t$a=;\x04*\xf8\x11\xd6\xec>*\x00]o\xc5\x8bo\x7f#%\'_\xe3.Z\x862\xd99R
\xd39c!\xab.H\x15{pp\xb3\x940y\xeeb\x93\xe0\xb1\xc8\xc7+3\xe7\x817\x01\xf1\x9eKr
~\xc2\xa7\xbb\xe3\xfc\xfa\xe5\xfck{\xe0\xe0\xb3\xech7\x1b\xef7P\xb4\xdc\x9b]\xc4
\xad\x86B\x8d\x98\xa5\x11\x9f\x02\xd4\xac\xdc\xdc\xaf/`\xcd\x0c\xc8\x83\x1f\xcd\
xc0bc\xbc9\x95r\x1d>\xdc\xab\xe4v\x1f\xde\xbb\xbd;\xd7\x1f\x18\xa8\xde\x90\x10OL
fW\x05\xdb\xab`|\xec\xd7\xbb\xac)\xa9\xf6\xf7\xdd\x96SH\xa9^\xe4WV\xdf\x84K\x99\
xe7l\x91\x88*\xa8\x87\xd9\xc7\xc7\xcd\x19\x11m\xb7\x96C\x81\xdf\xccc6\x9c\x12N\x
d2`e&1\x13\x92\x0e\xa1\xffX R\x99z*-S3%c$\x06\x88\xd6\x972X\x92\x95\x0b\xe7\xddj
\x11^$6I\x05\xfb5\xeao\xceg\xd2\x944\x87F\x9e\x0e\x05m\xf2\t\xf9\xdd*3\x97y\xb7\
xc20J\xaee%8\xba#e\x08\xa6\x97\x99$CT\xc7L\x9a\xe1\x16\xf8y\x86S\x00\xd3\xd0\xd2
\xbdn\xf2^?Y\x83[\x8b\xfcM\xec\xd2\xae\x90K\xd5\x13\xf35p\xab\x89/{\x16\xb0\x08\
x9e?\x19\xa8\x89\x16{\x1a7E\xf4\\\xbf\xdc\xf5]\xb8\\\xe5E\x984Y\xa1\xd0pg4\xcc\x
abH\x12\xbe\xd0\xba\xaeBP\xc2\x8c\x8c4\xa3\'\xaa2\t\xa8\x18<v\xa98D\x8f\xae\'\xb
8\xe5kT\x8e\xe2\xae\xc86\xc9\x0fHH\xc6\xdc\xda#\xc8W\xc2]0rY\x8d\xafNP.\xf4\x00\
xa8:\xfa\x87\xb66\xe9\xf9\xe0\xf3\xc6dN\xd0\xd3\xc0V\xdc2*I\trw\xb8\xf8\xda\x8d\
x8f\x0c\x1aW594IP\xb4j\xfd\xf7E\xcc\xb2R\x17\n\x99\xf76\x10\xc2\xf0\x0e\x8d\xae\
xf0\x0b\x1cU\xf1ZXd\xbe\xcc\x83H\x8d\xe7*\xae"\xaasP\xa3EGh\xec2\xa3T\xa8\xe5\xb
e\xfdA\xaf\xe9\xb0M\xf3Gw\xcb\x92\xcd\xb3\x89\x10\x05\xfe\xf2r\xfe\xd4\n\x15\x1a
N\x90\xbd\xde\x08\x90;j|\xfa\x0e\xe4@\xffA\x8cC\x9a\x058b\x00\xc2\xd0\x01\xc9\xa
7\x16td\xb2\xcc$\xe6w\x05\xa1\x85\x99\xe6\x92\xd1\x19C\xfe\xc4r\x95}v\x1f\x90\'#
+\x03\xce\x85K*E\x1c1r\x0f\x89\x8f\x7f~\xef\xeeu\x93\x15\x8e\xca\xac!\x97|i$\xa3
\xbf\x96\x9ad\xd9\xaa\xa0\xac^\x9e\x88y\xcf\xb9\xc9\xea\xa0,\xe2\x82b.Ri\x0f\x17
```

```
\x8d\xbe\x19}\xac\xabG8+\xc93D\x8a\xe6\xccP\x8f\x97T%1\x7f\xd2\nF\xaa\xa9\xa2\xd
2\xc4\x00\xd7\xe5\xc6\x19\xac\r\xf2\x03\xaf\x7f\xbb\xbc/m\xb5 \x10\x7f\x83\x95\x
859\xe9\xe4[{\xf9\x11\x95\x9f\xd5\x0c\x042a\xfc\x1f\x89\x8ex\t\x06E\r\xca\x96\x1
aL\xd1v\xcc\x059\xc0\x0ch\xa4\xfc\xe2\xcb\xab\x03\x02\xb76\n\xd4\xdeT\xaeskY\x93
o\x15TA\xcb>V \x96\t\'\xca\xd9\xcd\tz\x92\x80u\x14\xe7q\x1civ\xd3\x1f\xf1o\xf5^\
xf8YF\xac\x99u\r\x18\xef\xb7\xa6\x03D^7\x14\x08\xf5\x1d\x88\x80~\xe8\x10~\x8c\xc
9\xa5\x94\x0b#2\xf4f4\xc3\x92\xb7`\xc7\x12\xed\x9f\xe7\x93C\xff\x1dZh\x99\xc3\xc
fk\x01\xb9I\\\x03W\x17-g\xc12\x0b\xb7\t'
Secret Key:  b'\x15\x861\x84\xfcs\x83\xe5\x8f\xd7!/\xb3LE\xc5\xa2l%\xb6\xeb\xe9\
xe7A\x8ar\x90\x12\x807\xea\xec\xfb=\xee9\x92%+\xd0\xaay7i1Kw\xf7\\\xbc\x96\xdc\x
c5=N\x1fX\x86\xd4&N\x9d\xad\xfa\xa2\x8bf\xfd\xea\x17\x13n\xf8>\x80-\x93\x8a\xc3\
xf2P\xa9Q\xfe\xe9\nz\xfd$%A*\x0ca\x94\xa3\x96\xad\xea\x95\x1b)\xf7c\xac\xa8I5w\x
a3f\xb9\x082\x18\xbb\xd6{\xa9\xa6n\xb7A1\x0b\xeb\xc0\xa1\x1c\x91\x90"\x19\x84\xd
a\x02\x81\xc8\x02eD\x16Rc8\x06\xd8BP\x01\x00*\x818\x10\x02 \x0cXFB\xa3FJD\x82E$7
\x86L"\x01\x18\xb4\x0c\x8a\x06\r\t\x97qX\x08\x11\x90(m\x08@%\x037\x02\xd4\x86\x0
0X\x08\x06\xd38\x82\x92\xa8\x08\x04\xa1P\x11\xb1\x01\x90\xb8\x91\x18\x84-L\x14\x
82\xe1& \x19\xb8d\x121hZ\xb6\x91\x92$,\xa3\x90\x8c\x00\x98PX\x101Q20\x91\x960\x8
0\xc0\x84\x11C\x88LF\x8d$\x92)a\xb2%\xa3\x92I\x81\x000\x18\xb4\tc$0\xa4\x12@\x82
\x86\x0c\xc1\x04Q\x8a\xa8\x85\x99\x06\x04\x91\x16\x81\xdb2Q\x08@"\xc0\x94\x11\x1
8IAZ\x14"\x81\xb8IS\xc6\x91\xc8\x04\x8d\n\x830\x1a\xc4,L\x02i\x8a\x90h\x14\x03\x
8d\x08"fbBr\x8c@&@\x84\x05\x8c0D\x1a\x820a\x94,A6q\xd1 F\x12\xb0\x05 \x11PB0A\xd
2H.\x12\x05q\x02\xb9\x8c\x08\x80\x90\x81\xb2\x8cJ\x142\xe3D`Q\x92)\x105Q\x9a4\x8
c\x94\x16\x88\x1aFH\n\xa5E\xd3\x84\x89\xe0\x86q\xdc\xc6\r!\x87$!\x05\x92\xa4\xc6
L$\x91q\x88\x160\x19\xa7@\xa06I\x91\xb6\x89\xd1\x92\x00c\x80M\x12\x08R\xc4\xa4pb
\x82\x10\x11\x19\x8c\x9b\x90\x89T\x02DR\x18-\xdcB\t\xda\x00*\x93\x00)\x00\x89\x8
1\xe4\x00l\x1a&&\xd2B\x04\xd9\x92$"G\x0e\x19\xb8 \x0c\xb4\x81\x00\x00\x12\x1c\xc
5M\xcb\xc2E`\x18l\xdb\xa4\x11I@I\x98&\x89\x84@\x12\xe1\x06(\x14\tj\x82\xa4\x91\x
8a\x94\x01\xdb\x14b\n8\r\x03\x111\x1c\x17\x84\x83\x16h\x98\x06pB$q\x00\x91
\x1a\xc1, \xc2H\x81\x06\x89B\x10\x02JHeB\x96Q\x9b(B\x88@\x00\x19\x05&\x8bD\x86\x
0c\x12*A\xb8d\x18\xc6e\x1a\xb3e\x10\xa5%RH*\xcc\x18\x00\x01H&C6FL\x98%L\x08i\xa0
4d[\x04Eb\xb0\x89\x83(\r\xe1\x10I\x13\xa5\r\xc1\xc2\x10H\x10\r\xe0&\x8a\xa3\xa4\
x8c\x1a\x80-\x12\x86\x00\x00!\x02\x10\xc8I\x19\x000BFF\xe3\x02N\t\x81`A\x86I\x98
\xa6h\x0b\xa0Q\xcc\x96\x84\x93"\x12\x83D(D\xc6\x85$B\n\x8c$\x12\x94\xa2H\x126!\x
11\x90\x05\x03\x83\x11\x19\x00I\x02\x17\x01\x88\xc4D\xd4H$\x80\xb8L\x1bF!@\xc6!\
xc9\xb4M\x1a\'(\x02E&\x0c\x12J\xc8\x02m\x1222\x14!&\x12\x14\x84d\x00\x01\x8a\x96
@\n4)\x88\xb6e\x01\x10R\x8a0QZ\xa4\x8cD\x10p\xca\x12dL\x90p\x91" \xd0($\x84Fd\x1
20i\x8c\xc4\x8cd\x18L\xca\x16\x88\x08\x07\t\x11!,\x88\x94a!%n\xc3\xa8 \x01\xb5\r
\xd1\xa2$\xd2\xb2)\x14\xc3\x08\x11\x81\x84\x92\xb4\x80\xc1\x14\x82L\x94\x81\xd1\
xc0\x10\x08\x86\x00\xc8(D\xd36\x0e\x90$@\xdb\xc0p\n\xb30\xc2\x88MD\x86a\xa4\x02d
\x932\x01\x12\x99\x80\x1c\x10e[ \x90\x04BgRCpM\xd6\xd3\xeca^\xcdkx\x99R\xe9\x0fi
_\x8a\x06\xb5"\x05?\xe2{\x15]<"\xfdu4\x84naY7\'\xc0l;\x8f\xdbc\xf9g\x95Y:\xf2\xe
c\xb1\xb1\x83\'\x11\xd5&6\x08\xfd\x03wQ\x81\xb9p~\xfb\x97\xa9\xed\xb9\xc85"Pz\x8
ff\x19\xbb\xf1\xe0:\xa6x\\\xe1z\xe6\x00\xde\xbc\x11*5y\x03\x9a%\xced\xd3{\xff\xa
3\xdc\xe7:\xc4w\xa4\xda\x80X;\x8c\xe2\x02\xda\xc1\xba\xf9\xaf\xf4XM3,C\x1b\xdc\x
c35\r\xce\xd16\x1a\xa6\'^\x90\xcf\xf0i\xc9\xe2M\x8f7\x84\x86\xbe\t\xf9_\xea\x14\
xe4O\xec\xde5\x0c\xe4\x86\x1b\x15Y\x93{\xe6\xaa\xee\xf7\xaf\xb2g\x8a\x8aG)uJb\x9
a^\x16\x89;\x94\x84YM\xa6\xacw&;\x1d\x81{Q\xd3\x1b~\xdb\xe1\x16h0\x87\xfd\x8aA\x
b9u&E7n\xdb6\xeb\xd7c8\xceEs\xcb\x8b\xac\x96\x86t\x10a\xd0\t<j\xa4\x03H\xa0\xe2\
```

xba{$\xec\x8awb2>_i\x88D\xdf\xa2o\x89\'\xe1+/~\x81p\x8e\xc9\xb6\xe5\xdat \x91P\x
d0\xa0\xbbu\x82|\x8c\xf1\x95\xba?\x89G\xe8\x8e;{1\xac\xd2i\xfd\xf2&n\xb0\xc2)\xa
f\x07\xf3\xdb\xd3\xbf\x0e\x05\xd4\xed\xc9\xa2\x18f\x1d\x01e\xa7:\x08\xd4aU-\x13\
xaf\xd9\xd5\xcc\x9f\xecKq\x8d\xa4)X\x7f4.\xaf\xea\x9a\x93i\xc7\x10\xee@B[KH\xdf\
xb2\xf6[\xbd~5\xe5\x10\x06\\\x86tl\xe5\xf3\xf7h\x85\xaec\xa6\xb9\xe6Q\xb3\xd6\xa
5LN\xc6\x02:\x11`\xf8N\xf9\xb0\xd1\xf4\xb0\xf6\xf3\x90F\xc9M\xaa\xd21\x84_\x0e \
xc7f\x85\x059\xf5\x17L\xa7F\xe6\xca\x8f\xc8\xce\xc8\xf6\x96Hd9\xef\xddC\xa6\xa3\
xb0*E\x8d\xee\xed\xc2i\xfa\xcf\x02.\x03u\xd6\xa8\x01`\xcf\x8d\xef\xafa\xb3\xa5\x
f7x\xa4\xc7c.Y\x11\xef\xd6h\x18 \xa0\xe6v\xcf\xab\xf1l\xbe\x16\xcb\x07"1\xa2\x15
\x191\xa8f\xc1:\xdd9\xd7\x9a^\\Q\x07\x15y7\xfa\x02We\x9eV\xbc\x1b\xc1A\x1e4 Q\xb
3\xaa\xa6\x19\x96Z\x11-\xb9w&\x13CN~\xae\xcbuO\xfbn\xa3C\x84m\xb1\x84\x18iI\x19$
\xa6a\xd6\xc6<\xce\xf5\x04\x02\xe6n\x06$Vgs\x0f\x7f\xd5\x7fr\xe0g?\x9eYDs/z\x8b2
Du\xedC\xd1\xa09\xb4\x17Mio\x8d\xe4\xca\xe3E\x1cP\x80&\xcd\xc0}\xe7\x1d3ZF\xdd,x
\xa1\xc3\x93\xf7\x1a\xde=)\xb6v5\xbf\xa0\x99\xc3\xf4R\x80\xe8\x8e\xd0\x03#\xb1\'
\xcd2\x1a\x00\x06e\xe4\x02p\xa6a\x8a\xd8o\x14\x8b\x08\xae:\x0e\xb4y\x9f/\xc1\xe4
\xcf\xdbX\xf7\x0ce\xac&\xd2\x8fJ\xfd`\x8b\xa1\xc9\x90\xabN \xcd\xb1\xc0\xee\x1cJ
4\xb6\x1b\xba\x88r\xf6\x91m\x905\xe3\xe5(\xa4m/\xd0\xbf\xf43\xa0\xfcb\x0ea\xcf!\
x88\xb5\xc7L\xc7\xf3\xa0\xfa\xd2l/\x8e`\\A\xd8\xacz{\x9b\x0f\xc4\xc9F\x10\xc2\xf
f\x1c\xc08d\xbb\xd4\x16,Es\xd1\xc3t^\xee\xaa+\x15\xe0\x13j\x1e&\x13r\xc0\xae\xdd
\xaa\x91+P\x92~>\xab\xc7^\x9f\xc2\xd0\x9e6^\xd7$\x0f\x9b\x8e\xe9S\x00\xa5c=\xd7\
x96\x96b6\xba\xa2W\x18\x8f\xb6ZH\\k\xc6wH5\xefY\xaa\xe1\xa7\xcc\xc0V(\x19I_=NS<\
xf0\x97\xfewoM\xd7\x9d\x1d\x92\x8b\x9e\'\x82\xba:\x9e\xf8V\x96\xf4\xe8\x9a\xc5n\
xeb\x12\xa7\x08YT\xc1=\x0b\x00\xdd6>\xfcgo\xe0\xfe\x83\x1b\xa3b\xf7~\x19G\x0c\xf
9\x98pc\x0c\x11Ml:\xeb\x1e7\xf3Q(!\xd4\xfcW\xa2\xba\x16\xfa\x13\xfb!\x01\xbd\xd9
3\xf6i\xcf\xd1S0\x89\x9c\x12\xe1CZl\xd9\x80\xf4\xdb`,\x0ba?\xe1\xcf\xa9\xec\xdf\
x03\x01\x97\xf1\xe1\xd7\xf7\xa3\x8fM\xe0\x82\xb8\xb1\xe0\x02\xc5v@0\xaf\xd5\x01\
x00o\x91\x87\x90\tgd\xd9:\x16\xc4\xb3\xfb\xbf\xf6\xe8`\xa8p&m\x05r\xdf0\xf0\xf8.
\xab\x14^8\x17\xc5*\x1c2\x93\x10\x9asJ\xe6\x0c\xf2W\xc3\xd6\xdd\xcb\x15\x1f)\x1e
\x15\xe3\x93\xe0\xf15\x87\xcfs\xc6\xb5Z\x03=\x98\xa9\xca\x9b\x13\x0fq-\x0bp7\x98
\x842\xde"\xf6\xae\xfaBO\x08\xe0\x81\x9d\xd1\xb4\xbd:\x8c\x82\x99\xfe#\x03M\x99\
x8f\xb9;P(\x89\x11&\xf27\xfa\x834\x1de\x01|\x0fV\xfak*d\x1e\xf5\xf9\xa1\xe9\xc1\
x86S\xfah\x13\xfa\x19k\xb5\xea\x19]\xca3\x8e\xa0!-u\x04PVR(\xea\xf3kN6\xc5\x9b{\
xd0\xc9\x17\xdb\xaa\x14\xdf\xdc6\x8c\xb7_\x83\x83\x03\x02\x1e\xe3\xa4A\xbf\x02b{
F\xd0\xd4\xba?\x99\xa33\xc6\xa6\x8a\xff\xe5\xd1J\xe0\xdb;\rw\xea\xf8\x8b\xd6\xaf
&\x15\xdev9_\x11\x9d\x03p+\xfd\x95\x890\x85\xbb\x00<np\\\x04\x06/\x18)\xd7\xbb\x
c3\x17\x1e3\xd4p\x1f0\xba\xd9lf\x06\xfeb\xdc\xf5^\x1c\xcd\xd29\x02\x11\xd4`\xfaO
\x11\xa3n\xcf|\x17\xcar\x040\t\xa3\xff~\x03;\xa9\x14\xa2\x96\xe5\xd2\xf1sN\xaf\x
a7\x10K!\x82\xb9\xb8S$\xc8R\xe1\x1e\xe0\xda9U\xd3\x14\xeaMJ\x87\x9b$\xd0\xdc\xf9
\x90\xccD\xd7\xad\xcb"\x9e\xd0bo\x13\x17\x15\x7fb\n\xe66vI\x8d\x0eM\x8f]\xa2\x9b
\x85\x13YWVec\x88\x1b\r\xed\xd7\xa87\x93\x17\xf6$\x13\x89\xc4q%\xf6Rd\xa5\xa0XW@
\xdce_8\x11\xd2\xda!\xc2\x0f!\xbfc\xaa\xf2\x92\x86F\x0bf\x80u\xc0\x12\xe7\xecC E
j\xdb\x9d\x18`\xa0N\x9c\xea\x8a\xf7\xf2\xd6\xf38\x13J\xc2\xba@\xf7\xc4^\x82p\xe6
\'\xe1\x10\x0b`\xf0\xfd-&~N]\x8a|\x18\xe9\x88>\x90\x8cVe\xa4h\x93\xac\xb3\x0cs#\
xd9\x9cF\xc7\xcc\xcd\xd3=\x93W\xd4<U?\x96D\xd0\xbaHN\x0e\xdd4d\x08\x8d\xc0\x96\x
fe\xa2\xf3O:\x83\xec!"Bo\xa3\xc5\xa6\x9b\x81!\x80\xa0\x03\x91\xad\xc7[\xa1\xda\x
a8bw\x8bC\x0e\xc3F\xc5\x15\xf2\xde0\x94\xc9\\z\xa0D\xb5\xfaZn\xed\x92\xd2\xc21P\
x14r\t\xcd\x7fV\xba\x88\xf22dna\xc7J\xdf\x1c\xac\xd0\xbb|A`T?\xbc\xb4\x13\xe7\x0
7e!\xd1\xf3\x1c\xbf\x87\xb9\x8f\xc5\x82%\x821\x80\x97\x96\xc0]-\xebDq[\xee\xd21\

```
xff\xfb4\x1a4\xee\xdd:\'\xc9\x99|O\x0c\x18^\xf8\\hx\x00\xbe\x99\xa2\xbai\xb2&Z\x
8e\x15\xc4\x96\xad\x89I&\xbdd\xca~\x16\xdb\x8b
XP\xb5\xe18\xb9=@]\x9e\rFp\xea\x88\xb9+\xa4\xed\x18\xa1\xa9\xf8\xe2\xe9\xb1'
```


1.1.5 Assinatura

[327]:
```
print("SIGN")
sig = dilithium.sign(sk, b"Hello World")

print("Assinaura (sigma): ", sig)
```

```
SIGN
```

```
Assinaura (sigma):  b'\x15\x8bg\xb8\xf4\xe45Va\xdd\x87\xecY\x1f&q\x07\xf4\xf3JP\
x0eJN%\xe6u\xe05\xaaR\x82\xf5\xa1\x8a\xf5\x15\x89\xdfw\xf2\x1d\xed|\xfaC\x80N\xf
c\xae1\x19e\xd2#\xb4\xadx\xb9y\xef\x90\x86\x90k\x838\xeceS\x1a\x93\xd0N\xfd\x86\
xde\xf1s\x94j{G\xe2~\xc9\xe6_\xcc\x12tv\x96\xe5\xc1\xa3\xe0\x14\xd7\x89\x1a\xbc\
xacz\xd9\x0c\xe8.!\xe7\x8b\xbb+\xce\xba\x9e\xa3Z0\xc4/\xab`;\xb3\xa9-\xc2\xd4\t\
t\xabL\x1d\x9f\x83u\xbfuJ\xd3H\n\x08\x06\xf5\x8f\xe0\xf1\x07<i\xb8Q\x16\xd1A<\xc
8\x99\x13\x9c\x12\xd8\xf9\x0bn\xb5\x04\xd6\x84Kw\xa4\x7f\xf1g_\xa1\x0f\x95Mt\x0b
63I\xc8\xb8\x15\x12\x95\xf2a\x12\x05\xecy\x81"\r\xae\xda\r|=\x01)W\x10\xc2\xd8\x
ec\x97cn\x8d\xf3P\x0b\xd1G\x97(#\xa3\xb5T\xb9\x82`\xe8O\xea\x7fX\tr<\x07\xa16\x1
d\x88!\xfc+\xf5\x82\xd6\x00\xe8\xda\x08\x89b\xc0\x19\xf1\x1f\x19\xd8\xd1\x0e\x1f
\x98\xa7b\xa1[\xd7\xfd\xe1\x1a\x97oOs*@Ca\xf2\xdb\xd5\xc1\xcc%&\xacjB)Mf\xcem\xf
a\xb1\x02\x94\x92+)\xdc\xfau\xe9}\xb3\xf0\xf2\xc5\xff\xd8\x98_\xb2I\x07\r\xae\xf
9)@\x84\xf4\xc1\x8aU\xceUV\x1a\xffZmNs4\x1b\xe0G\xe5\x17\xe7\x85\x01\xd3$\x87\x1
e\xee\x1f\xa8cK\xc2o3\x97\xf3\xd90\x98\xabIG\xec\xb1v1R\xdf1uk\xe3#\x94\xdd=E\x8
e\x98\x8b\xec\x0c\x1c8\x0eo\x05eD\x11FX\xfd!\xd6\xa6Z\xbb\xaf\xcf;B\x1d\xad\x0c\
xa4\xf2\xd4\xdff\xbb\xf2gcy\xeb\xb1\xe3\xdf1\xc81\x08>N\r\xd0\xb7\xb1\x0fz\xfc\x
18U-l\xa9\xc8\x7f([a\xf6\xb5~\xbb\xa8P\xf4\x12#\xaeB$*\x8b\xb2_u-\xa0*m&\xa7\xc5
\xca\xfe\xfe-^\xa4\x0ea\xb6B\\9\xc5g\xb2\xfb\x9b\x1d\x91\r\x8f\x8d\'\xc5\xedO\xb
b\xf4\'}\xed\x9d\xfc{\x15\x8c\xe72\xbfxSa\xda\xd6!)y\xbe\xb6\xf3]\x02\xac\xbf6\x
a4\xda\xfbx|\xa2\x9bW@\x95^\x1e\x9c{\x93\xec\xc9 V"]o\x0c\xe1\xcbh+e\x9b~\xc3f\x
05\x9d\xe2\x91\x19P\xc2\x1c\x1cy\x06e\xc2(\xb9S8\\X\x1e\x0b\xca\xaf;\x9d\xa8\x00
\xc2\xd7a&\xfa\x95\xe8Q\xe3EHJ\x1az%\x84\xce\xbeO\x16\x00\x80F\xc1\xf1V\xee\x8d1
\x12{\xa5\x9a\x93\x1d~\xb7\x19*s\x1d5\xb9k\xda\x0b\xa4\x0c-Z|\x9b\xbf\x9e\xe8X<\
x92\x8b\x85\xfc]L9Of\xebA\x1a\xf7{1\x12\x93\xaa9B\x00:\xd4i\x1cPk\x0f/\xe1\xe9\n
1\xe8\xd3\'2\x8b\xe9T!\xe6J\x91Ft\xb0\xcc\xcd8!2^?r:0\xeac?2\x82t\x91\x94\x89\x9
8\x17^{\xc2[\x078\xd3Jp\xceT\xc7\xdb\xe1\x96\'i7\xb9\x94\x84i<;\x9b@\x17\xe92\xc
f\x92\x14\xa8\x1c\xb4\xc1P,,_3\x03\x0en\x82\x8fkc\x1f\xc7\n\xba%\xee\xbe\x0f\xfa
\xde\xach\xfb\x91\x89 ua\xf8%\xaf\xadOm\x89\xbe\xa7\x18\x85\xa28\xdb\t\x1e2a*X\x
ab\xdcbO\x90\x85\x8esA~\x84d\xff!\x0b\xc1\x91^2\xe8\xc5\x0693\x8f\x7f\xe2G\xce\x
f0E`O\x82\x03\xbc\xe5\xe8\xa0}m0\xb0^h\x92\x19.Dk\n\xed\xb0\x98\xeab\x8e\xacM\xc
d\xde\x814\x90G\xf6[\x9a\xd3\xe2j\xfaZP\xd2\xc0\x8a\xc7\xcc\x1b\xaa\xfbzi~\xf62k
\xb0\xb02;?\x19i{\xe9\x83\xaf\xa4\x1e9\xe3\xaf\xae\x13\xc1\xc0u\xa9\xf9!\xaaT\x9
a\x91rn?\t/\xac\x88J\x18\x06%\xd9\xdd\xe4\xdf\xff\xd9\xbf"K;\xa59\xb6#}\xe4\xfd\
x9d\x87\n\xad\xb1\x16\xb1\xb6\x8a\xe8\x80>\x1b=\xf6\x8c\x91\xc8o\xcb\xb0\xe1\xbc
\xf40\x1b\xb2\x10\xe9\xd2\xd9z\xc8\xa2\xd2\xf2a\xd3#p\xc5\xb6/K\xc1;\xbe\x11\xe3
\xe51\xb9\x99\x04"y\xfed*\xe6\xd4U\r\xc9\xce^\x8e\xb9$ \\tre\x94\xd6"\xa8\xf7\xd
```

```
7\xf3-\x99pS2\xf0\x175f:d\x9a\x0e\x06\x15\xe5\xdb\x91g\x00\xa4\x14\x7f0\xc8\xa8\
'\xea\xac\xd4\xbap\xd8 1\x97H\x94\xa5+\xbd\x90\xd1{\xb7\xb8\xf0\x89\xe7*\xf8F\x9
3\x85\xa9\xea\xcf\xceq\xc5m\x9c\xcb\x01\x81\xf89\xd8\xc4r\xc9L\x12\x0erkY\x92H\x
e8\x86Y\xa4(m\xc5\x1bDH\x7fje\xd4\xb8\xbb\x08\xdd\xf7\xc2\x90^9m\xca\xc1\x7f\xbb
\xed\t\x19\x00\x8b\xf1(\xa3\x06=]v\xb9g\x11t\x97\xf0\x85v\xc3\xbbV5\xf3Y*\xe1\x1
3q5\xebi\xb5|s:\xc3\xca\x9d\x897\x81\x18\xd9:_&\x07,\x7f\x02\xf0z\x11\x01\xbe\xd
30D\xda\xb5\x87\xc6O&\x08V\xf7~\x91\xb3\x0b\x93\xe7zMd\x1d\xc7.I\x83\x89\xc4pO\x
87\xc6N\x01R#%\xfd{\xa7CB\x93\xbc\x94#-!A)\xed\xbb\x127L"]\xbf\x13w\xd7\x18\x88P
\xac%;\x0b\x8c\x94\xb3\xed9\xed#\x06EjV\xca!f\x14\x8b\x96\x8a\xca\x17f7\x8d\xe9\
x8b\xd0\xbb\xd1uc-\x8d\x96\x08+\xe1Y^\xf3\x17\x1fS\xe1\x1d\x06\xe3-\xa3\xd0B\xdc
\xf5\x9c\xe7q\xfe\x83\xf2D\x98M\xb2\xaa\x03\xfdf\xc5g!\xc0\x10+[?G3m\x83\x05v\xa
4\xbf\xb9\xed\x13k\x01\xfe8\xf6}\xa6\xb9k\xcb\x81\xa5\xa0\xf35,mS\x8c\x8d\xe95\x
85\xaa\x7f3\xd6\xfcY^\x82\xa3G\x073\xc7]\x03\xcd\xa9<!wg\x92\xf1e\x9d\x07G\xec\x
cen/Z\x12%^\x1cR\xac?p\x834\xcd-\xe02\x07v`7\x7fB<\xe6?\x01\xc4s\xa2\x1a0;5R\xb1
O]\x9e\x9c\xed\x01lq\x890\xc1\x07\xcf3\n\x87T\xbd\x10,\xb3\xfbf\xedr\xecDNJ0\xd8
\x82\xda/5\xa5A\xd9\xb3\xf6\t\xdd\x8e\xcd6\xaa\x02#MZ\x05\xe4xHpz\x87\x8f6/\xaaD
\x90\r\x14&\xd6]\xa2\xdb!\x00\xd7$3\x0f\xfe\x1c\x16,W\xbad\xf5\xa9\x07\x98\x16B\
xd2\x1e2Q\x1e$z\x8c\xaa\xdaj\xb3\x08\xc6\xbc)\x00-\x0b\xcf\x10/:$\x1a\x95\xe9\xb
b]\xf3n\xc2\x16\xa2\xc6\xc4\xdf\xb2\xc3\x0b\x15\xf9{o\xfdEd\xdd\xf6\xd4\xae\x83\
xe1\xaa\xbdIU\x03\x12W\xc7\xd6\xc4\xa0\x13\xe89\xcf\x96/\xc1\xf9\x95-\xa5\xfc\xf
a\x08\xa7\x08$\xe21R\xc5\xf4\x8b{G7\x0b\n\xf4<\t\x1bl\x97\xaa\x8f\xdb\xbf\x825\x
fc\xcdpB\xa4\x85,\x8e\x17\x8d\xa2\xaf\x9f\xc1\',\xed\xed\xfb\xf9\x83\xd9\xb52\x1
4\x83q\x87\xa4\x94\x88\xb8.\xd9\xbe-\xc2B\x86\xa9\x1djKo\x98\xa6#dj\x0b(\x15\xf5
sI\xf3\xc9\xb0[\x15\x00\xbck\x8f\x98\xdf\xe6\x1d9\\4\x1b\xd2\xc4\xba%\x0f\xb3m\x
876\xb3O\xe8|\xda\xff\xbe\x18\x04Ay\xa1\xfc\x08N\xf9\to\xaa)\x90\xb7\xdaD\xc1f\x
00\xe5\xcd\xc2u\x00\xb0\\\x92\xf8\xa5\x8cH\xb6\xac\xa5\xf9l\xcbC\xcaH:\n+W\x15\x
94\xc6\'\xeeI\x07\xa3\n\xf7\x10\x9aA\xea\x9f#\xc1\xd6\xc9\xca\x0cH\x9e\xd1AJ\xd2
=\xa6\x1b\xb8\xdb\xf6I\xc0\x8d\x1ek\xf7C\xf9\x10q\xcb\xbf\x0e\xd7?4\x0c\x85\xee?
\xf4Q\x93\x07\xe2\xf1\xa3\xcdC\xb8\xaaY\xa7\xca\x91\xdc\xfa\xa7gL=\xa4\xf0=\x08\
xd0\x1a\xe9\xf5\xcd(\xdeX\xe3\x8cw\xaa\xb6_J\x0eW\xf5\xa2\x8a\x05\x1anp\x11\xa2\
x8f\xb8\xc8\x87&\x17~\xb1P\xc3\xa2\xd8y\xf8H4\xe1\xac0\xacir\x9fi[\xeb\x16{\x90\
xef\x99\x84\x0f\xb2r\xfe0>\x0e\xc4MppB\x7f\xf33Q>U\x17\xd9\x00\xe3\xbaS\xe0\' At
\xdc\xd3\xa7jk\xa3\xf9\xd7\xfa[\x81\xe6\x12\x15\xe1\xdcnfH\xb9\xaf\xe5S\xf6\xa3\
xce\xf6\xf9T\xe9\x94\xddi\xd5\xce\xbe\x19\x0eA\xda0\xdc+\xbe\x92\xd1\x91]_l\x83\
xc8K\xb3\xef!M<q\xa1[,\x8e\x04\xb0=\x8aHd\nOrN\xb1\xa07^"\x9b8\x1e[:\x1a\x06<\x8
c\xbb\xacY\x10i\xee\xf3No\x19\x04v?J\xcd#X~\\\xbc\x8a\xd2\xa6GnF\x979\x0f\x8f\xa
6r\xdch\xee\xd4\xf9{L\xbc\'\xcdy\x12{s\x82\xe1\x16\xfd\x9a\xbe\xdb|\xd4\xb0\x14\
x02\x1c\x86\x02\x13\xbdS=\xe9\xb8S\xc2P\xbcT\xc9\xff\xccu\xe4\xbf\xe4,/\x83v8\xd
4\xa1@Ab\x06\xe1\xfa\xb7\x16eC\xae\x8b\xfa\xf5\x7f\xec3\x01\xa2\x8bT\xe2\x15\xa2
\x8844N\x9b\xa6\xd4\x04B)\xc6\x05d2\xf8\xdf#\xdbD\x0b\xfc\xc6N,(\x18\xc2M1\x81\x
ce2\xcd\x13F\xc1\x8b\x8dC\x9d\x84\x9d\xaa\xa2\xcc7U\xdfv\x9a\x0c\xc0@\x9a\x05\x8
e\x1e\xda\x8d\xe6%\xe4\x82\xab\x0ff\t9\x8e\x9f5BY\xa6\x89H\xae%Tu\xfa9E\x06M\xab
m\x91\xf3\xf6&;%\x9f\xf0c\xa9>b\x11/\xb8\x80\xcc\xbdf@+/\x9c\xf4u\x1a\xc2\xc6\x8
a\x9f\xba\xf2]\x7f\xa6\xed\xc03\x08\xba~\xe9\xfe\xab\x84\x965\xa1\x1f1\x1e_3\xc3
\xafbc\xb5NA\t\xedm"S\x06x\xde\x87\xe6\xac\xc6\xce\xe1G\xadC\x00\x89\xb2\xb32\xb
1\xd8-\x7f\xca\xc8G\x80\xc8\x0f\xc9r\x8b\xcf._Y\xbf\x18V\xd5bmD?\xc0j\xa3l\x8b\x
f5e\xfdt\x80q\xa2g\x89\xfa\xddB\x85\xd3qNqR\xec\xa2I\x07\xff\x00\x00\x00\x00\x00
\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00
```

```
\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00
\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00
\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x00\x01\x01'
```

1.1.6 Verificação

[328]:
```python
print("VERIFY")
# dilithium.Verify(pk, b"Hello World", sig)
```

```
VERIFY
```