# Dilithium

June 7, 2021

## 1 Implementação Dilithium

Abaixo temos a implementação do esquema de assinaturas pós-quantico Dilithium conforme as especificações da terceira submissão para o concurso do NIST.

[46]:
```
import os
import random

from cryptography.hazmat.primitives import hashes
```

**Parâmetros do Dilithium** Abaixo temos um dos conjuntos pré definidos de parâmetros para o Dilithium.

[16]:
```
'''
q = 8380417
d = 13
tau = 39
challenge netropy = 192
gamma1 = 2^17
gamma2 = 95232
(k, l) = (4, 4)
eta = 2
beta = 78
omega = 80
reps = 4.25
'''
```

[16]:
```
'\nq = 8380417\nd = 13\nGAMMA = 39\nchallenge netropy = 192\ngamma1 =
2^17\ngamma2 = 95232\n(k, l) = (4, 4)\neta = 2\nbeta = 78\nomega = 80\nreps =
4.25\n'
```

### 1.0.1 Geração do Par de Chaves

[8]:
```
def Gen():
    #01
    zeta = os.urandom(32)

    #02
```

```
    (rho, rhol, K) = H02(zeta)

    #03
    A = ExpandA(rho)
    Acirc = NTTmat(A)

    #04
    (s1, s2) = ExpandS(rhol)

    #05
    As1 = INTT(Acirc*NTT(s1))
    t = As1 + s2

    #06
    (t1, t0) = Power2Roundq(t,13)

    #07
    tr = H07(rho + t1)

    #08
    pk = (rho, t1)
    sk = (rho, K, tr, s1, s2, t0)
    return pk, sk
```

### 1.0.2 Assinatura

[2]:
```
def Sign(sk, M):
    sk = (rho, K, tr, s1, s2, t0)

    #09
    A = ExpandA(rho)
    Acirc = NTTmat(A)

    #10
    mu = H1012(tr + M)

    #11
    kappa = 0
    z = 0; h = 0

    #12
    rhol = H1012(K + mu)

    #13
    s1circ = NTT(s1)
    s2circ = NTT(s2)
    t0circ = NTT(t0)
```

```
    while z == 0 & h == 0:

        #14
        y = ExpandMask(rhol, kappa)

        #15
        w = INTT(Acirc * NTT(y))

        #16
        w1 = HighBitsq(w, 2 * 95232)

        #17
        ctil = H(mu, w1)

        #18
        c = SampleInBall(ctil)
        ccirc = NTT(c)

        #19
        cs1 = INTT(ccirc*s1circ)
        z = y + cs1

        #20
        cs2 = INTT(ccirc*s2circ)
        r0 = LowBitsq(w-cs2, 2 * 95232)

        #21
        if size_of_elements_bige(z, 2^17 - 78) | size_of_elements_bige(e0,␣
↪95232 - 78):
            z = 0
            h = 0
        #22
        else:
            #23
            ct0 = INTT(ccirc * t0circ)
            h = MakeHintq(-ct0, w - cs2 + ct0, 2 * 95232)
            #24
            if size_of_elements_bige(ct0 , 95232) | number_of_1s_big(h, 80):
                z = 0
                h = 0
        #25
        kappa = kappa + l

    #26
    sigma = (ctil, z, h)
    return sigma
```

### 1.0.3 Verificação

[1]:
```
def Verify(pk, M, sigma):
    pk = (rho, t1)
    sigma = (ctil, z, h)
    #27
    A = ExpandA(rho)
    Acirc = NTTmat(A)

    #28
    mu = H(H(rho + t1) + M)

    #29
    c = SampleInBall(ctil)

    #30
    wl1 = UseHintq(h, INTT((Acirc * NTT(z)) - (NTT(c) * NTT(t1*(2^13)))))

    #31
    return size_of_elements_low(z, 2^17 - 78) & ctil == H(mu + wl1) &␣
↪number_of_1s_lowe(h, 80)
```

### 1.0.4 Execução de um Teste

[ ]:
```
pk, sk = Gen()

M = os.urandom(32)

sigma = Sign(sk, M)

done = Verify(pk, M, sigma)

print(done)
```

### 1.0.5 Função H da linha 2

[37]:
```
def H02(x):
    digest = hashes.Hash(hashes.SHAKE256(int(128)))
    digest.update(x)
    buffer = digest.finalize()
    return buffer[:32], buffer[32:96], buffer[-32:]
```

### 1.0.6 Função H da linha 7

[26]:
```
def H07(x):
    digest = hashes.Hash(hashes.SHAKE256(int(32)))
    digest.update(x)
    return digest.finalize()
```

### 1.0.7 Função H das linhas 10 e 12

[ ]:
```
def H1012(x):
    digest = hashes.Hash(hashes.SHAKE256(int(64)))
    digest.update(x)
    return digest.finalize()
```

### 1.0.8 ExpandA

[ ]:
```
def ExpandA(x):
    # TODO
    return 0
```

### 1.0.9 ExpandS

[ ]:
```
def ExpandS(x):
    # TODO
    return 0
```

### 1.0.10 ExpandMask

[ ]:
```
def ExpandMask(x1, x2):
    # TODO
    return 0
```

### 1.0.11 NTT

NTT conforme a borboleta CT.

[34]:
```
def NTT(a):
    psi = []
    n = len(a)

    t = b

    m = 1
    while m < n:
        t = t//2
```

```
        for i in range(m):
            j1 = 2 * i * t

            j2 = j1 + t - 1

            # TODO
            S = 'sadsadasdsadasda'

            for j in range(j1, j2+1):
                U = a[j]
                V = a[j + t] * S

                a[j] = (U + V) % 8380417

                a[j + t] = (U - V) % 8380417

        m = m*2

    return a
```

### 1.0.12 INTT

INTT conforme a borboleta GS.

[5]:
```
def INTT(a):
    n = len(a)

    t = 1

    m = n
    while m > 1:
        j1 = 0

        h = m//2

        for i in range(h):
            j2 = j1 + t - 1

            # TODO
            S = 'assadsdasdas'

            for j in range(j1, j2+1):
                U = a[j]
                V = a[j+t]
                a[j] = (U + V) % 8380417
                a[j+t] = ((U - V) * S) % 8380417
```

```
            j1 = j1 + 2 * t

        t = 2 * t
        m = m//2

    for j in range(n):
        # TODO ou //
        a[j] = (a[j] / (n)) % q

    return a
```

[5]: 1

### 1.0.13 NTTmat

NTT aplicado a cada elemento de uma matriz.

[ ]:
```
def NTTmat(M):
    for i in range(len(M)):
        M[i] = NTT(M[i])

    return M
```

### 1.0.14 INTTmat

INTT aplicado a cada elemento de uma matriz.

[ ]:
```
def INTTmat(M):
    for i in range(len(M)):
        M[i] = INTT(M[i])

    return M
```

### 1.0.15 SampleInBall

[55]:
```
def SampleInBall(rho):
    random.seed(a=rho, version=2)
    c = []
    for i in range(256):
        c.append(0)

    for i in range(256 - 39, 256):
        j = random.randint(0, i)
        s = random.randint(0, 1)
        c[i] = c[j]
        c[j] = (-1)^s
```

```
    return c
```

### 1.0.16 Power2Roundq

[70]:
```
def Power2Roundq(r, d):
    r = r % 8380417

    r0 = r % (2^d)
    r0 = r0 - (2^(d-1))

    return ((r - r0)//(2^d)), r0
```

### 1.0.17 MakeHintq

[ ]:
```
def MakeHintq(z, r, alpha):
    r1 = HighBitsq(r, alpha)
    v1 = HighBitsq(r + z, alpha)

    return r1 != v1
```

### 1.0.18 UseHintq

[ ]:
```
def UseHintq(h, r, alpha):
    m = (8380417 - 1)//alpha

    (r1, r0) = Decomposeq(r, alpha)

    if h == 1 & r0 > 0:
        return (r + 1) % m

    if h == 1 & r0 <= 0:
        return (r - 1) % m

    return r1
```

### 1.0.19 HighBitsq

[ ]:
```
def HighBitsq(r, alpha):
    (r1, r0) = Decomposeq(r, alpha)
    return r1
```

### 1.0.20 LowBitsq

[ ]:
```
def LowBitsq(r, alpha):
    (r1, r0) = Decomposeq(r, alpha)
    return r0
```

### 1.0.21 Decomposeq

[ ]:
```
def Decomposeq(r, alpha):
    r = r % 8380417

    r0 = r % alpha
    r0 = r0 - (alpha//2)

    if r - r0 == 8380417 - 1:
        r1 = 0
        r0 = r0 - 1
    else:
        r1 = (r - r0)//alpha

    return (r1, r0)
```

### 1.0.22 Tamanho de Elementos

Cálculo de ||w||∞ e verificação se é menor que uma condição.

[89]:
```
def size_of_elements_low(w, cond):
    x = w % 8380417
    x = x - (8380417//2)
    if x < 0:
        x = -x

    return x < cond
```

### 1.0.23 Tamanho de Elementos

Cálculo de ||w||∞ e verificação se é maior ou igual que uma condição.

[90]:
```
def size_of_elements_bige(w, cond):
    x = w % 8380417
    x = x - (8380417//2)
    if x < 0:
        x = -x

    return x >= cond
```

### 1.0.24 Número de 1's

Cálculo do número de 1's num número em binário e verificação se é maior que uma condição.

[91]:
```
def number_of_1s_big(w, cond):
    counter = 0
    aux = w
    while aux > 0:
        if aux % 2:
            counter += 1
        aux = aux // 2

    return counter > cond
```

### 1.0.25 Número de 1's

Cálculo do número de 1's num número em binário e verificação se é menor ou igual que uma condição.

[92]:
```
def number_of_1s_lowe(w, cond):
    counter = 0
    aux = w
    while aux > 0:
        if aux % 2:
            counter += 1
        aux = aux // 2

    return counter <= cond
```