

A series of thin, black, intersecting lines of various lengths and orientations, creating a complex, abstract geometric pattern in the upper left portion of the page.

UMA IMPLEMENTAÇÃO DIDÁTICA DO ALGORÍTIMO KYBER

Rafael Hass - 103852

AGENDA

Quem sou eu

Introdução

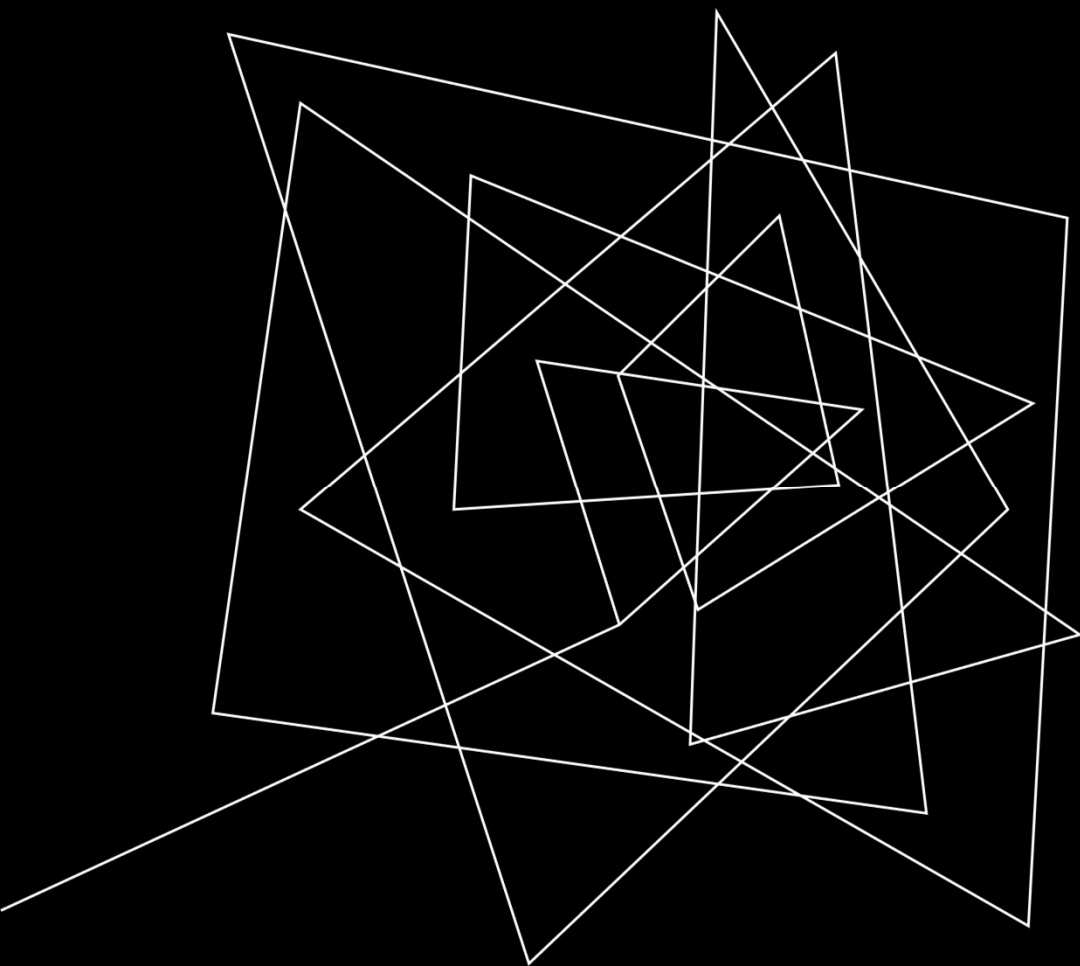
Implementação PKE

Implementação KEM

Resumo matemático

QUEM SOU EU

- Formado em Licenciatura em Matemática em 2014
- Trabalhando com segurança da informação desde 2016
- Atualmente diretor de SOC e Threat Intelligence no banco BTG Pactual.



INTRODUÇÃO

Algoritmo Kyber

2017

O algoritmo CRYSTAL-Kyber é proposto como parte da iniciativa de submissão pós-quântica do NIST (National Institute of Standards and Technology) para algoritmos criptográficos.

2020

O NIST seleciona CRYSTAL-Kyber como um dos finalistas para o terceiro round da competição PQC.

2021

A terceira rodada da competição PQC está em andamento, com testes e avaliações mais aprofundados dos algoritmos finalistas, incluindo CRYSTAL-Kyber.

O NIST realiza um workshop público para discutir os algoritmos finalistas e obter feedback da comunidade criptográfica.

2022

O NIST anuncia os algoritmos pós-quânticos selecionados como padrões, incluindo o anúncio de CRYSTAL-Kyber como um dos algoritmos escolhidos.

LINHA DO TEMPO



ALGUMAS CARACTERÍSTICAS DO KYBER

Resistência algoritmos pós quânticos

Resistência a
algoritmos pós
quânticos conhecidos

Baseado em reticulados

Grade multidimensional
de pontos, onde a
segurança é baseada na
dificuldade de resolver
problemas matemáticos
complexos relacionados a
essa estrutura.

MLWE

LWE trabalha com vetores de
inteiros
Ring LWE trabalha com
polinômios
MLWE trabalha com vetores
de polinômios inteiros módulo
 q

SVP

(Shortest Vector
Problem), que envolve
encontrar o vetor mais
curto em um reticulado.



ALGUMAS CARACTERÍSTICAS DO KYBER

Kyber PKE - IND-CPA

O IND-CPA (Indistinguishability under Chosen-Plaintext Attack) é um modelo de ataque onde um adversário tem a capacidade de escolher textos-claros e criptografá-los usando o esquema criptográfico que está sendo avaliado. Em seguida, o adversário recebe as cifras correspondentes e pode realizar operações e análises com base nessas cifras.

Kyber KEM - IND-CCA2

O IND-CCA2 (Indistinguishability under Chosen-Ciphertext Attack) é um modelo de ataque mais forte em comparação ao IND-CPA. Nesse modelo, um adversário tem a capacidade de escolher pares de texto-claro e texto-cifrado, e também pode realizar operações e análises com base nessas informações.

Gerar chaves

s = Matriz de dimensões $k \times 1$ de polinômios de tamanho n
 A = Matriz de dimensões $k \times k$ de polinômios de tamanho n
 e = Matriz de dimensões $k \times 1$ de polinômios de tamanho n
Retornamos $PK = (t = A * s + e, A)$ $SK = s$

Encriptar

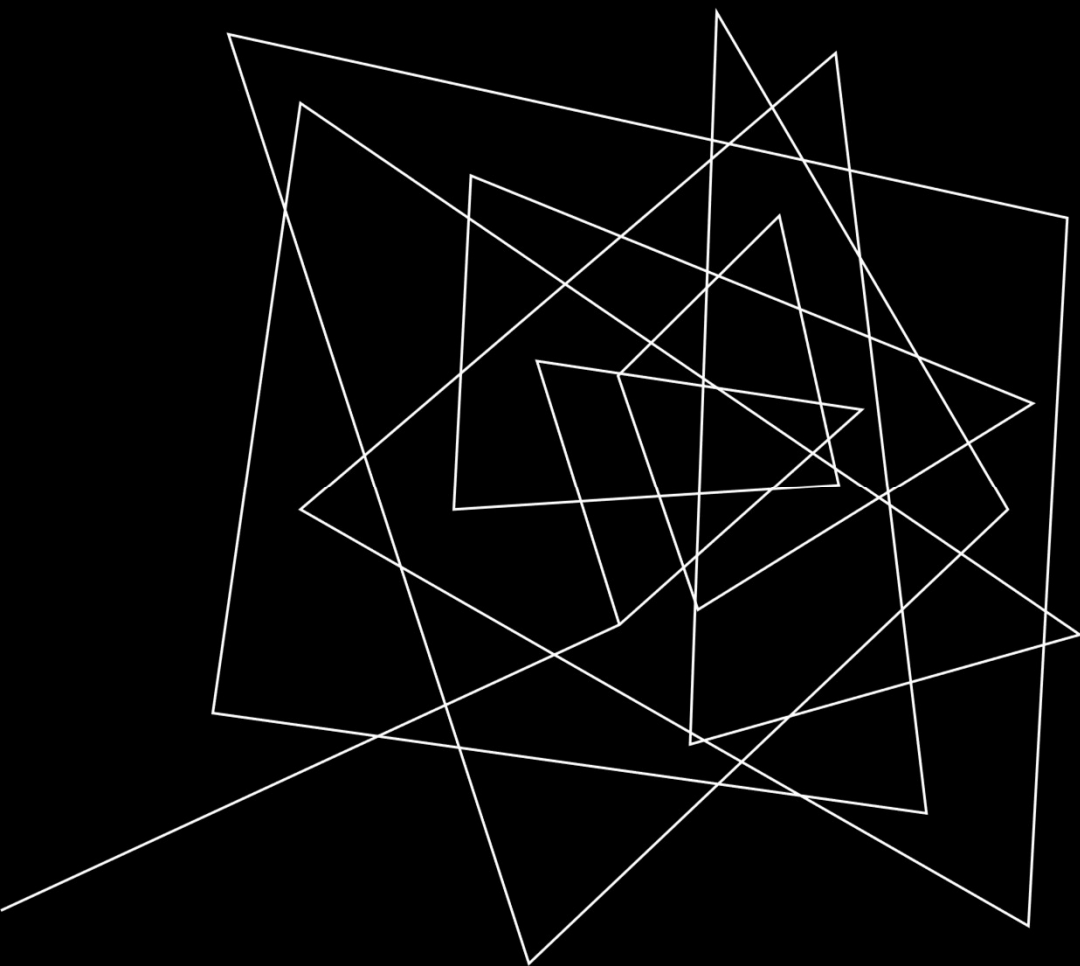
m = mensagem em binário
 r = Matriz de dimensões $k \times 1$ de polinômios de tamanho n
 $e1$ = Matriz de dimensões $k \times 1$ de polinômios de tamanho n
 $e2$ = Matriz de dimensões 1×1 de polinômios de tamanho n
Retornamos $u = A^T * r + e1$ e $v = t^T * r + e2 + m_u$

Decriptar

$$m^* = v - s^T * u = m + e2 + e^t * r + s^T * e1$$

E para achar o m , basta verificar para cada item do m^* , se está mais próximo de $q/2$ ou de $(0$ ou $q)$

LINHA DO TEMPO



IMPLEMENTAÇÃO DIDÁTICA

Algoritmo Kyber PKE



O QUE É

Vou apresentar uma implementação simplificada do algoritmo CRYSTALS-Kyber, para exemplificar de forma didática alguns dos passos na execução, mantendo a lógica da implementação original, mas ignorando algumas etapas como compressão e descompressão, dado que não são relevantes para o entendimento do Kyber.

Essa não é uma implementação que deve ser usada em qualquer aplicação do mundo real.

FUNÇÕES AUXILIARES USADAS

```
def createRandomMatrixOfArrayOfIntLimited(dimX: int, dimY: int, dimArrays: int, minValue: int, maxValue: int):  
    matrix = np.empty((dimX, dimY), dtype=np.ndarray) # Create the matrix  
    # Generate random values and assign them to each array  
    for i in range(dimX):  
        for j in range(dimY):  
            random_values = np.random.randint(low=minValue, high=maxValue, size=dimArrays)  
            matrix[i, j] = random_values  
    return matrix
```

Gerar matriz de vetores com números aleatórios

```
def setPolynomialRing(A: np.ndarray):  
    rows = A.shape[0]  
    columns = A.shape[1]  
    matrix = np.zeros((rows, columns), dtype=np.ndarray)  
  
    for i in range(rows):  
        for j in range(columns):  
            div, d = np.polynomial.polynomial.polydiv(A[i][j], cofdiv)  
            matrix[i][j] = d % q  
  
    return matrix
```

Fazer os polinômios módulo $x^{256} + 1$ e módulo q

FUNÇÕES AUXILIARES USADAS

```
def sum_arrays(A: np.ndarray, B: np.ndarray):  
  
    if A.shape != B.shape:  
        raise Exception("Dimensions incompatible:" + str( A.shape)+'x'+str(B.shape))  
    rows = A.shape[0]  
    columns = A.shape[1]  
    matrix = np.zeros((rows, columns), dtype=np.ndarray)  
    for i in range(rows):  
        for j in range(columns):  
            poly_sum = np.polynomial.polynomial.polyadd(A[i][j], B[i][j])  
            matrix[i][j] = poly_sum  
  
    return(setPolynomialRing(matrix))
```

Fazer a soma de matriz de polinômios

```
def mult_matrix_polinomyals(A: np.ndarray, B: np.ndarray):  
  
    rows = A.shape[0]  
    columns = B.shape[1]  
    matrix = np.zeros((rows, columns), dtype=np.ndarray)  
    if A.shape[0] % B.shape[1] != 0:  
        raise Exception("Dimensions incompatible:" + str( A.shape[0])+'x'+str(B.shape[1]))  
  
    # Perform matrix multiplication  
    for i in range(A.shape[0]):  
        for j in range(B.shape[1]):  
            for k in range(B.shape[0]):  
                resultMult = np.polynomial.polynomial.polymul(A[i][k], B[k][j]) # Line A x Column B  
                matrix[i, j] = np.polynomial.polynomial.polyadd(matrix[i,j], resultMult)  
  
    return(setPolynomialRing(matrix))
```

Fazer a multiplicação de matriz de polinômios

PARÂMETROS A SEREM UTILIZADOS

Table 1: Parameter sets for KYBER

	n	k	q	η_1	η_2	(d_u, d_v)	δ
KYBER512	256	2	3329	3	2	(10, 4)	2^{-139}
KYBER768	256	3	3329	2	2	(10, 4)	2^{-164}
KYBER1024	256	4	3329	2	2	(11, 5)	2^{-174}

```
import numpy as np
import math

n = 256 # 256
q = 3329 # 3329
k = 2
n1 = 3
n2 = 2
cofdiv = np.zeros(n+1, dtype=int)
cofdiv[0] = 1
cofdiv[-1] = 1
```

- n representa o tamanho em bits, da mensagem a ser encapsulada.
- K representa o número de polinômios por vetor.
- q representa um primo pequeno que satisfaz $n|(q-1)$. É utilizado também para fazer o módulo das multiplicações de polinômios.
- n_1, n_2, d_u, d_v representa um balanço entre segurança, tamanho do texto cifrado e chance de erro. Em nossa implementação, não usaremos d_u, d_v pois são utilizados na compressão e descompressão das chaves e textos cifrados.
- δ representa a chance de erro.

FUNÇÃO DE GERAR CHAVES

```
def generateKey():
```

```
    s = createRandomMatrixOfArrayOfIntLimited(k,1,n,-n1,n1+1)
    A = createRandomMatrixOfArrayOfIntLimited(k,k,n,-q,q)
    e = createRandomMatrixOfArrayOfIntLimited(k,1,n,-n1,n1+1)

    multA_S= mult_matrix_polinomyals(A,s)
    t = sum_arrays(multA_S, e)
    return ((A,t),s)
```

Algorithm 4 KYBER.CPAPKE.KeyGen(): key generation

Output: Secret key $sk \in \mathcal{B}^{12 \cdot k \cdot n/8}$

Output: Public key $pk \in \mathcal{B}^{12 \cdot k \cdot n/8 + 32}$

```
1:  $d \leftarrow \mathcal{B}^{32}$ 
2:  $(\rho, \sigma) := G(d)$ 
3:  $N := 0$ 
4: for  $i$  from 0 to  $k-1$  do
5:   for  $j$  from 0 to  $k-1$  do
6:      $A[i][j] := \text{Parse}(\text{XOF}(\rho, j, i))$ 
7:   end for
8: end for
9: for  $i$  from 0 to  $k-1$  do
10:   $s[i] := \text{CBD}_{\eta_1}(\text{PRF}(\sigma, N))$ 
11:   $N := N + 1$ 
12: end for
13: for  $i$  from 0 to  $k-1$  do
14:   $e[i] := \text{CBD}_{\eta_1}(\text{PRF}(\sigma, N))$ 
15:   $N := N + 1$ 
16: end for
17:  $\hat{s} := \text{NTT}(s)$ 
18:  $\hat{e} := \text{NTT}(e)$ 
19:  $\hat{t} := \hat{A} \circ \hat{s} + \hat{e}$ 
20:  $pk := (\text{Encode}_{12}(\hat{t} \bmod^+ q) \| \rho)$ 
21:  $sk := \text{Encode}_{12}(\hat{s} \bmod^+ q)$ 
22: return  $(pk, sk)$ 
```

▷ Generate matrix $\hat{A} \in R_q^{k \times k}$ in NTT domain

▷ Sample $s \in R_q^k$ from B_{η_1}

▷ Sample $e \in R_q^k$ from B_{η_1}

▷ $pk := As + e$
▷ $sk := s$

FUNÇÃO DE ENCRYPTAR

```
def encryption(arrayMessageB, SK):
    A = SK[0]
    t = SK[1]
    r = createRandomMatrixOfArrayOfIntLimited(k,1,n,-n1,n1+1)
    e1 = createRandomMatrixOfArrayOfIntLimited(k,1,n,-n2,n2+1)
    e2 = createRandomMatrixOfArrayOfIntLimited(1,1,n,-n2,n2+1)
    m = math.ceil(q/2)*arrayMessageB

    #u = A.T + e1
    mult_At_r = mult_matrix_polinomyals(A.T,r)
    u = sum_arrays(mult_At_r, e1)

    #v = t.T*r + e2 + m
    mult_tt_r = mult_matrix_polinomyals(t.T,r)

    sumE2M = sum_arrays(e2, m)
    v = sum_arrays(mult_tt_r,sumE2M)

    return (u,v)
```

Algorithm 5 KYBER.CPAPKE.Enc(pk, m, r): encryption

Input: Public key $pk \in \mathcal{B}^{12 \cdot k \cdot n/8 + 32}$
Input: Message $m \in \mathcal{B}^{32}$
Input: Random coins $r \in \mathcal{B}^{32}$
Output: Ciphertext $c \in \mathcal{B}^{d_u \cdot k \cdot n/8 + d_v \cdot n/8}$

- 1: $N := 0$
- 2: $\hat{t} := \text{Decode}_{12}(pk)$
- 3: $\rho := pk + 12 \cdot k \cdot n/8$
- 4: **for** i from 0 to $k-1$ **do** ▷ Generate matrix $\hat{A} \in R_q^{k \times k}$ in NTT domain
- 5: **for** j from 0 to $k-1$ **do**
- 6: $\hat{A}^T[i][j] := \text{Parse}(\text{XOF}(\rho, i, j))$
- 7: **end for**
- 8: **end for**
- 9: **for** i from 0 to $k-1$ **do** ▷ Sample $\mathbf{r} \in R_q^k$ from B_{η_1}
- 10: $\mathbf{r}[i] := \text{CBD}_{\eta_1}(\text{PRF}(r, N))$
- 11: $N := N + 1$
- 12: **end for**
- 13: **for** i from 0 to $k-1$ **do** ▷ Sample $\mathbf{e}_1 \in R_q^k$ from B_{η_2}
- 14: $\mathbf{e}_1[i] := \text{CBD}_{\eta_2}(\text{PRF}(r, N))$
- 15: $N := N + 1$
- 16: **end for**
- 17: $\mathbf{e}_2 := \text{CBD}_{\eta_2}(\text{PRF}(r, N))$ ▷ Sample $\mathbf{e}_2 \in R_q^k$ from B_{η_2}
- 18: $\hat{\mathbf{r}} := \text{NTT}(\mathbf{r})$
- 19: $\mathbf{u} := \text{NTT}^{-1}(\hat{A}^T \circ \hat{\mathbf{r}}) + \mathbf{e}_1$ ▷ $\mathbf{u} := \mathbf{A}^T \mathbf{r} + \mathbf{e}_1$
- 20: $v := \text{NTT}^{-1}(\hat{t}^T \circ \hat{\mathbf{r}}) + \mathbf{e}_2 + \text{Decompress}_q(\text{Decode}_1(m), 1)$ ▷ $v := \mathbf{t}^T \mathbf{r} + \mathbf{e}_2 + \text{Decompress}_q(m, 1)$
- 21: $c_1 := \text{Encode}_{d_u}(\text{Compress}_q(\mathbf{u}, d_u))$
- 22: $c_2 := \text{Encode}_{d_v}(\text{Compress}_q(v, d_v))$
- 23: **return** $c = (c_1 \| c_2)$ ▷ $c := (\text{Compress}_q(\mathbf{u}, d_u), \text{Compress}_q(v, d_v))$

FUNÇÃO DE DECRYPTAR

```
def decription(s, cm):  
    u = cm[0]  
    v = cm[1]  
  
    #mn = v - s_t*u  
    mult_st_u = mult_matrix_polinomyals(s.T,u)  
  
    mn = sum_arrays(v, -mult_st_u)  
  
    # print('mn: ' + str(mn))  
  
    resultado = []  
    for item in mn[0][0]:  
        if abs(item - math.ceil(q/2)) < math.ceil(q/4):  
            resultado.append(1)  
        else:  
            resultado.append(0)  
  
    while len(resultado) < n:  
        resultado.append(0)  
  
    return [resultado]
```

Algorithm 6 KYBER.CPAPKE.Dec(sk, c): decryption

Input: Secret key $sk \in \mathcal{B}^{12 \cdot k \cdot n/8}$

Input: Ciphertext $c \in \mathcal{B}^{d_u \cdot k \cdot n/8 + d_v \cdot n/8}$

Output: Message $m \in \mathcal{B}^{32}$

1: $\mathbf{u} := \text{Decompress}_q(\text{Decode}_{d_u}(c), d_u)$

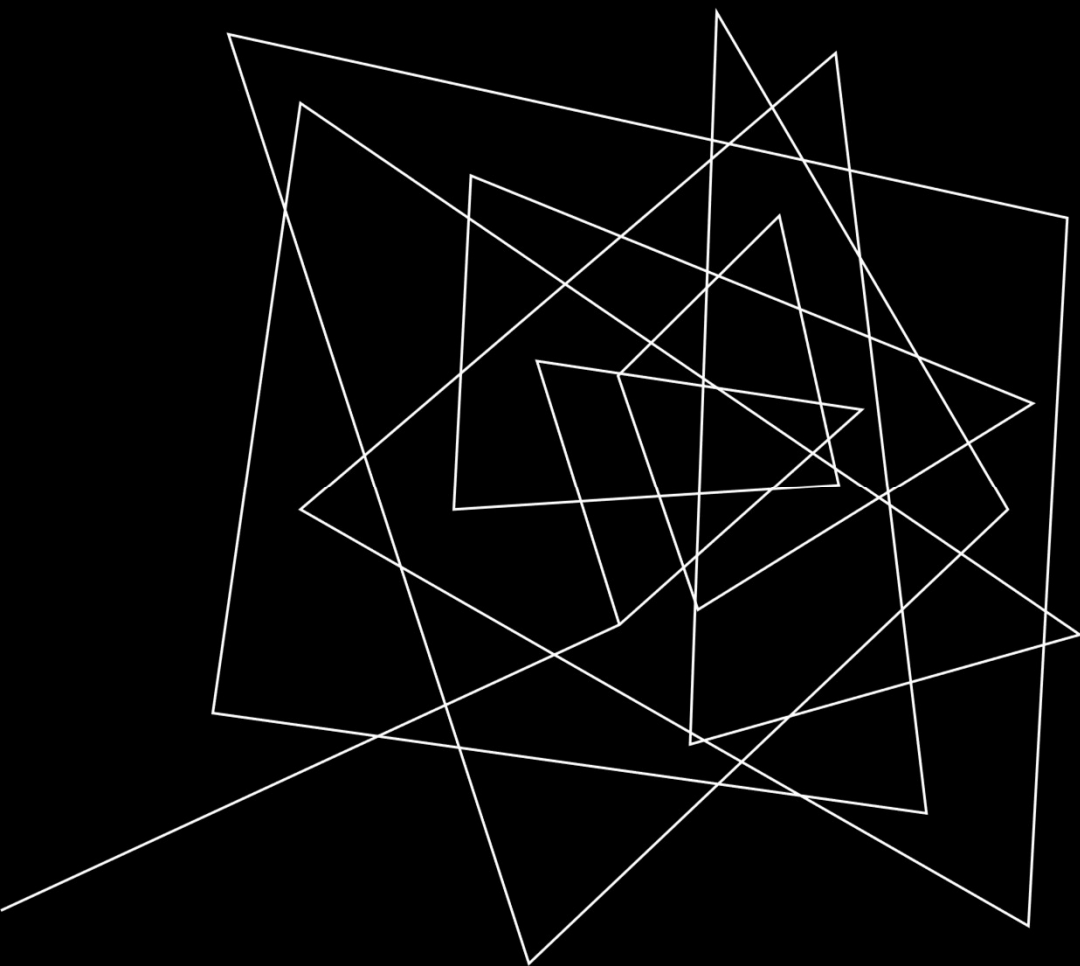
2: $\mathbf{v} := \text{Decompress}_q(\text{Decode}_{d_v}(c + d_u \cdot k \cdot n/8), d_v)$

3: $\hat{\mathbf{s}} := \text{Decode}_{12}(sk)$

4: $\mathbf{m} := \text{Encode}_1(\text{Compress}_q(\mathbf{v} - \text{NTT}^{-1}(\hat{\mathbf{s}}^T \circ \text{NTT}(\mathbf{u})), 1))$

$\triangleright m := \text{Compress}_q(\mathbf{v} - \mathbf{s}^T \mathbf{u}, 1)$

5: **return** \mathbf{m}



IMPLEMENTAÇÃO DIDÁTICA

Algoritmo Kyber KEM

FUNÇÃO DE GERAR CHAVES

```
def KEM_KeyGen():  
    return generateKey()
```

Algorithm 7 KYBER.CCAKEM.KeyGen()

Output: Public key $pk \in \mathcal{B}^{12 \cdot k \cdot n / 8 + 32}$

Output: Secret key $sk \in \mathcal{B}^{24 \cdot k \cdot n / 8 + 96}$

1: $z \leftarrow \mathcal{B}^{32}$

2: $(pk, sk') := \text{KYBER.CPAPKE.KeyGen}()$

3: $sk := (sk' || pk || H(pk) || z)$

4: **return** (pk, sk)

FUNÇÃO DE ENCRYPTAR

```
def KEM_Encrypt(pk):  
    m = createRandomMatrixOfArrayOfIntLimited(1,1,n,0,2)  
    ct = encryption(m, pk)  
    return ct,m
```

Algorithm 8 $\text{KYBER.CCAKEM.Enc}(pk)$

Input: Public key $pk \in \mathcal{B}^{12 \cdot k \cdot n/8 + 32}$

Output: Ciphertext $c \in \mathcal{B}^{d_u \cdot k \cdot n/8 + d_v \cdot n/8}$

Output: Shared key $K \in \mathcal{B}^*$

1: $m \leftarrow \mathcal{B}^{32}$

2: $m \leftarrow H(m)$

▷ Do not send output of system RNG

3: $(\tilde{K}, r) := G(m \| H(pk))$

4: $c := \text{KYBER.CPAPKE.Enc}(pk, m, r)$

5: $K := \text{KDF}(\tilde{K} \| H(c))$

6: **return** (c, K)

FUNÇÃO DE DECRYPTAR

```
def KEM_Decrypt(sk, ct):  
    return decryption(sk, ct)
```

Algorithm 9 $\text{KYBER.CCAKEM.Dec}(c, sk)$

Input: Ciphertext $c \in \mathcal{B}^{d_u \cdot k \cdot n/8 + d_v \cdot n/8}$

Input: Secret key $sk \in \mathcal{B}^{24 \cdot k \cdot n/8 + 96}$

Output: Shared key $K \in \mathcal{B}^*$

```
1:  $pk := sk + 12 \cdot k \cdot n/8$   
2:  $h := sk + 24 \cdot k \cdot n/8 + 32 \in \mathcal{B}^{32}$   
3:  $z := sk + 24 \cdot k \cdot n/8 + 64$   
4:  $m' := \text{KYBER.CPAPKE.Dec}(s, (u, v))$   
5:  $(\tilde{K}', r') := G(m' \| h)$   
6:  $c' := \text{KYBER.CPAPKE.Enc}(pk, m', r')$   
7: if  $c = c'$  then  
8:   return  $K := \text{KDF}(\tilde{K}' \| H(c))$   
9: else  
10:  return  $K := \text{KDF}(z \| H(c))$   
11: end if  
12: return  $K$ 
```



OBRIGADO

Rafael Hass

rafaelhass.92@gmail.com

<https://github.com/rafaelhass/MO421>