

Rafael Hass - 103852

AGENDA

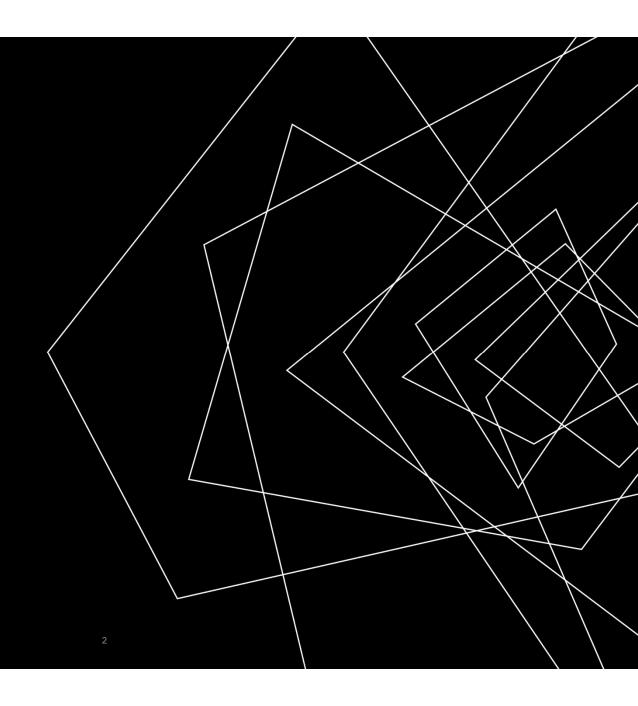
Quem sou eu

Introdução

Implementação PKE

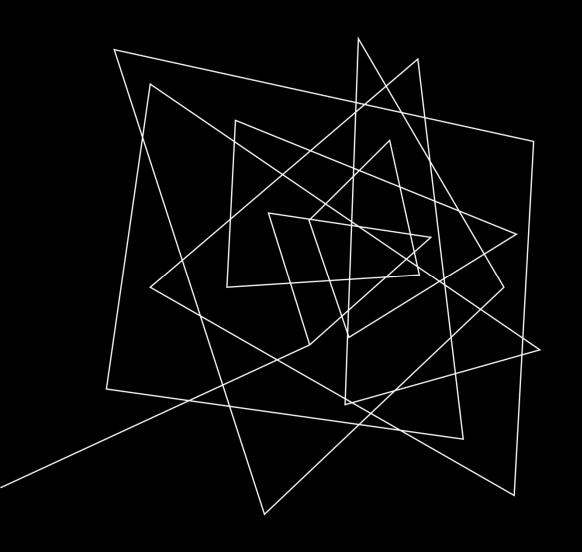
Implementação KEM

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 Inteligence no banco BTG Pactual.



INTRODUÇÃO

Algorítimo 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 algorítimos 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

SVP

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

q

ALGUMAS CARACTERÍSTICAS DO KYBER

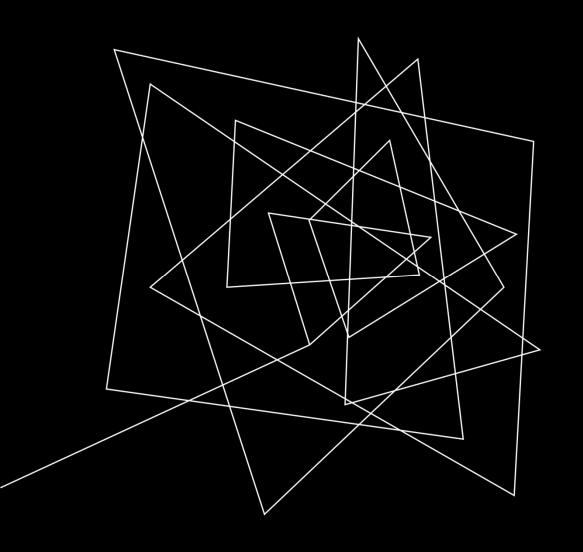
Kyber PKE -IND-CPA

Kyber KEM - IND-CCA2

O IND-CPA (Indistinguishability under Chosen-Plaintext Attack) é um modelo de ataque onde um adversário tem a capacidade de escolher textosclaros 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. 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 kx1 de polinômios de tamanho n A = Matriz de dimensões kxk de polinômios de tamanho n e = Matriz de dimensões kx1 de polinômios de tamanho n Retornamos PK = (t = A*s + e,A) SK = s Encriptar m = mensagem em binário r = Matriz de dimensões kx1 de polinômios de tamanho n e1 = Matriz de dimensões kx1 de polinômios de tamanho n e2 = Matriz de dimensões 1x1 de polinômios de tamanho n Retornamos $u = A^T * r + e1$ e $v = t^T * r + e2 + m_u$ $m^* = v - s^T * u = m + e^2 + e^t * r + s^T * e^1$ Decriptar 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 createRandomMatrixOfArraysofIntLimited(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[0]):
        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
 23329
 32
 2(10,4)
 2^{-139}

 KYBER768
 256
 33329
 22
 (10,4)
 2^{-164}

 KYBER1024
 256
 43329
 22
 (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 polinônios.
- 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 = createRandomMatrixOfArraysofIntLimited(k,1,n,-n1,n1+1) A = createRandomMatrixOfArraysofIntLimited(k,k,n,-q,q) e = createRandomMatrixOfArraysofIntLimited(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 ← B<sup>32</sup>
 2: (\rho, \sigma) := G(d)
 3: N := 0
                                                                                                     \triangleright Generate matrix \hat{\mathbf{A}} \in R_a^{k \times k} in NTT domain
 4: for i from 0 to k-1 do
           for j from 0 to k-1 do
                  \mathbf{A}[i][j] \coloneqq \mathsf{Parse}(\mathsf{XOF}(\rho, j, i))
           end for
 8: end for
 9: for i from 0 to k-1 do
                                                                                                                                     \triangleright Sample \mathbf{s} \in R_q^k from B_{\eta_1}
           s[i] := \mathsf{CBD}_{\eta_1}(\mathsf{PRF}(\sigma, N))
           N := N + 1
12: end for
                                                                                                                                    \triangleright Sample \mathbf{e} \in R_a^k from B_{\eta_1}
13: for i from 0 to k-1 do
           e[i] := CBD_n, (PRF(\sigma, N))
           N := N + 1
16: end for
17: \hat{\mathbf{s}} := \mathsf{NTT}(\mathbf{s})
18: \hat{\mathbf{e}} := \mathsf{NTT}(\mathbf{e})
19: \hat{\mathbf{t}} := \hat{\mathbf{A}} \circ \hat{\mathbf{s}} + \hat{\mathbf{e}}
20: pk := (\mathsf{Encode}_{12}(\hat{\mathbf{t}} \bmod^+ q) || \rho)
                                                                                                                                                        \triangleright pk := \mathbf{As} + \mathbf{e}
21: sk := \mathsf{Encode}_{12}(\hat{\mathbf{s}} \bmod^+ q)
                                                                                                                                                                  \triangleright sk := s
22: return (pk, sk)
```

FUNÇÃO DE ENCRIPTAR

```
def encription(arrayMessageB, SK):

A = SK[0]
t = SK[1]
r = createRandomMatrixOfArraysofIntLimited(k,1,n,-n1,n1+1)
e1 = createRandomMatrixOfArraysofIntLimited(k,1,n,-n2,n2+1)
e2 = createRandomMatrixOfArraysofIntLimited(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{\mathbf{t}} := \mathsf{Decode}_{12}(pk)
 3: \rho := pk + 12 \cdot k \cdot n/8
 4: for i from 0 to k-1 do
                                                                                                      \triangleright Generate matrix \hat{\mathbf{A}} \in R_a^{k \times k} in NTT domain
           for j from 0 to k-1 do
                  \hat{\mathbf{A}}^T[i][j] := \mathsf{Parse}(\mathsf{XOF}(\rho, i, j))
            end for
 8: end for
 9: for i from 0 to k-1 do
                                                                                                                                     \triangleright Sample \mathbf{r} \in R_n^k from B_{n_1}
           \mathbf{r}[i] := \mathsf{CBD}_{\eta_i}(\mathsf{PRF}(r, N))
          N := N + 1
12: end for
13: for i from 0 to k-1 do
                                                                                                                                   \triangleright Sample \mathbf{e}_1 \in R_a^k from B_{n_2}
           e_1[i] := CBD_{\eta_2}(PRF(r, N))
          N := N + 1
16: end for
17: e_2 := CBD_{n_2}(PRF(r, N))
                                                                                                                                    \triangleright Sample e_2 \in R_q from B_{n_2}
18: \hat{\mathbf{r}} \coloneqq \mathsf{NTT}(\mathbf{r})
19: \mathbf{u} := \mathsf{NTT}^{-1}(\hat{\mathbf{A}}^T \circ \hat{\mathbf{r}}) + \mathbf{e}_1
                                                                                                                                                      \triangleright \mathbf{u} := \mathbf{A}^T \mathbf{r} + \mathbf{e}_1
20: v := \mathsf{NTT}^{-1}(\hat{\mathbf{t}}^T \circ \hat{\mathbf{r}}) + e_2 + \mathsf{Decompress}_q(\mathsf{Decode}_1(m), 1)
                                                                                                                      \triangleright v := \mathbf{t}^T \mathbf{r} + e_2 + \mathsf{Decompress}_q(m, 1)
21: c_1 := \mathsf{Encode}_{d_u}(\mathsf{Compress}_q(\mathbf{u}, d_u))
22: c_2 := \mathsf{Encode}_{d_v}(\mathsf{Compress}_q(v, d_v))
                                                                                                           \triangleright c := (\mathsf{Compress}_q(\mathbf{u}, d_u), \mathsf{Compress}_q(v, d_v))
23: return c = (c_1 || c_2)
```

FUNÇÃO DE DECRIPTAR

```
def decription(s, cm):
    u = cm[0]
    v = cm[1]

#mn = v - s_t*u
    mult_st_u = mult_matrix_polinomyals(s.T,u)

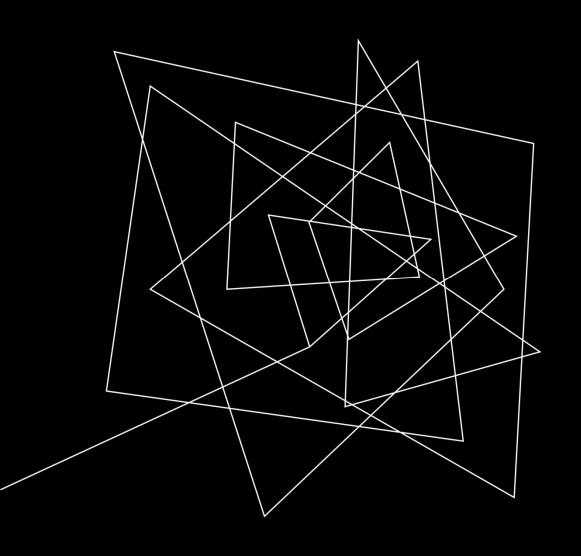
mn = sum_arrays(v, -mult_st_u)

# 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]</pre>
```



IMPLEMENTAÇÃO DIDÁTICA

Algoritmo Kyber KEM



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 ← B³²
- 2: (pk, sk') := Kyber.CPAPKE.KeyGen()3: sk := (sk' || pk || H(pk) || z)
- 4: return (pk, sk)

return generateKey()

```
FUNÇÃO DE ENCRIPTAR
def KEM_Encrypt(pk):
   m = createRandomMatrixOfArraysofIntLimited(1,1,n,0,2)
   ct = encription(m, pk)
   return ct,m
```

Algorithm 8 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 ← B³²
- 2: $m \leftarrow H(m)$
- 3: $(\bar{K}, r) := G(m || H(pk))$
- 4: c := KYBER.CPAPKE.Enc(pk, m, r)
- 5: $K := \mathsf{KDF}(\bar{K} || \mathsf{H}(c))$
- 6: return (c, K)

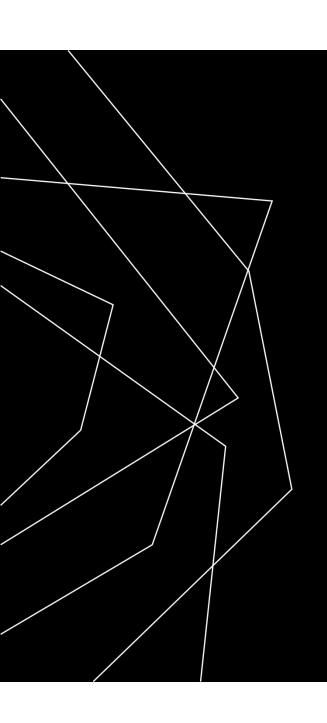
Do not send output of system RNG

FUNÇÃO DE DECRIPTAR Algorithm 9 Kyber.CCAKEM.Dec Input: Ciphertext $c \in \mathcal{B}^{d_u \cdot k \cdot n/8 + d_u \cdot n}$ Input: Secret key $sk \in \mathcal{B}^{24 \cdot k \cdot n/8 + 96}$

```
Algorithm 9 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}(\mathbf{s}, (\mathbf{u}, v))
 5: (\bar{K}', r') := G(m'||h)
 6: c' := \text{KYBER.CPAPKE.Enc}(pk, m', r')
 7: if c = c' then
       return K := KDF(\bar{K}' || H(c))
 9: else
        return K := KDF(z||H(c))
11: end if
12: return K
```

def KEM_Decrypt(sk, ct):

return decription(sk, ct)



OBRIGADO

Rafael Hass

rafaelhass.92@gmail.com