Estruturas Criptográficas PG53886, Ivo Miguel Alves Ribeiro A95323, Henrique Ribeiro Fernandes

A key-encapsulation mechanism (or KEM) is a set of algorithms that, under certain conditions, can be used by two parties to established using a KEM can then be used with symmetric-key cryptographic algorithms to perform basic tasks in secure communications, such as encryption and authentication. This standard specifes a key-encapsulation mechanism called ML-KEM is related to the computational diffculty of the so-called Module Learning with Errorsproblem. At present, ML-KEM is believed to be secure even against adversaries who possess a quantum computer Neste trabalho pretende-se implementar em Sagemath um protótipo deste standard parametrizado de acordo com as variantes sugeridas na norma (512, 768 e 1024 bits de segurança) Hash Functions

import secrets

2. Em Agosto de 2023 a NIST publicou um draf da norma FIPS203 para um Key Encapsulation Mechanism (KEM) derivado dos algoritmos KYBER. O preâmbulo do "draft"

In [ ]: import hashlib def Random\_32\_bytes():

return secrets.token\_bytes(32) def H(s): return hashlib.sha3\_256(s).digest()

def J(s): return hashlib.shake\_256(s).digest(32) def G(s): return hashlib.sha3\_512(s).digest() **def** XOF(x, i, j): return (hashlib.shake\_128(x + bytes([i, j])).digest(256\*3))

def PRF(data, b, eta): return hashlib.shake\_256(data + bytes([b])).digest(64 \* eta) def polyADD(a, b, Q): return [(x + y) % Q for x, y in zip(a, b)]def polySUB(a, b, Q): return [(x - y) % Q for x, y in zip(a, b)]Algoritmo 2 BitsToBytes e Algoritmo 3 BytesToBits

In [ ]: def BitsToBytes(bits): assert(len(bits) % 8 == 0) return bytes( sum(bits[i + j] << j for j in range(8))</pre> for i in range(0, len(bits), 8)

def BytesToBits(data) bits = [] **for** word **in** data:

for i in range(8) bits.append((word >> i) & 1) **return** bits Algoritmo 5 ByteEncode e 6 ByteDecode

In [ ]: def ByteEncode(F, d): assert(len(F) == 256)bits = [] for a in F: for i in range(d): bits.append((a  $\gg$  i) & 1) return BitsToBytes(bits)

def ByteDecode(B, d): bits = BytesToBits(B) return [sum(bits[i \* d + j] << j for j in range(d)) for i in range(256)]</pre> def Compress(x, d, Q): return [(((n \* 2\*\*d) + Q // 2 ) // Q) % (2\*\*d) for n in x] def Decompress(x, d, Q): return [(((n \* Q) + 2\*\*(d-1)) // 2\*\*d) % Q for n in x]Algoritmo 6 SampleNTT In [ ]: def SampleNTT(B, Q): res = [] i = 0while len(res) < 256:</pre> a, b, c = B[i:i+3]i**+=**3  $d1 = ((b \& 0xf) << 8) \mid a$ d2 = c << 4 | b >> 4 **if** d1 < Q:

res.append(d1) if d2 < Q and len(res) < 256: res.append(d2) **return** res Algoritmo 7 SamplePolyCBC assert(len(B) == 64 \* eta)bits = BytesToBits(B) f = [] for i in range(256): x = sum(bits[2\*i\*eta+j] for j in range(eta)) f.append((x - y) % Q)return f Algoritmo 8 NTT

In [ ]: def SamplePolyCBC(B, Q, eta): In [ ]: def bitrev7(n): return int(f"{n:07b}"[::-1], 2) def NTT(f\_in, Q):  $f_{out} = f_{in.copy()}$ k = 1 for log2len in range(7, 0, -1): length = 2\*\*log2lenfor start in range(0, 256, 2 \* length): zeta = pow(17, bitrev7(k), Q)k += 1 for j in range(start, start + length): t = (zeta \* f\_out[j + length]) % Q  $f_{out[j + length]} = (f_{out[j]} - t) % Q$  $f_{out}[j] = (f_{out}[j] + t) % Q$ **return** f\_out Algoritmo 9 NTT-1

In [ ]: def NTT\_inv(f\_in, Q):  $f_{out} = f_{in.copy()}$ k = 127for log2len in range(1, 8): length = 2\*\*log2lenfor start in range(0, 256, 2 \* length): zeta = pow(17, bitrev7(k), Q)for j in range(start, start + length):  $t = f_out[j]$  $f_{out}[j] = (t + f_{out}[j + length]) % Q$ **for** i **in** range(256):  $f_{out}[i] = (f_{out}[i] * 3303) % Q$ return f\_out Algoritmo 11 NTT mult In [ ]: **def** NTT\_mult(a, b, Q): c = []for i in range(128): a0, a1 = a[2 \* i: 2 \* i + 2]b0, b1 = b[2 \* i: 2 \* i + 2]c.append((a0 \* b0 + a1 \* b1 \* pow(17, 2\*bitrev7(i)+1, Q)) % Q) c.append((a0 \* b1 + a1 \* b0) % Q) **return** c K-PKE Component Scheme

def \_\_init\_\_(self, n, q, k, n1, n2, du, dv):

rho, sigma = ghash[:32], ghash[32:]

for j in range(self.k):

for j in range(self.k):

In [ ]: class K\_PKE:

self.n = nself.q = qself.k = kself.n1 = n1self.n2 = n2self.du = duself.dv = dv

def keygen(self):

ghash = G(d)

row = []

 $a_hat = []$ 

 $t_hat = []$ 

d = Random\_32\_bytes()

for i in range(self.k):

a\_hat.append(row)

for i in range(self.k):  $sum = e_hat[i]$ 

t\_hat.append(sum)

ek\_pke = b"".join(ByteEncode(s, 12) for s in t\_hat) + rho dk\_pke = b"".join(ByteEncode(s, 12) for s in s\_hat) return ek\_pke, dk\_pke def PKE\_Encrypt(self, ek\_pke, msg, r):  $t_hat = [ByteDecode(ek_pke[i*384:(i+1)*384], 12)$  for i in range(self.k)]  $rho = ek_pke[-32:]$  $a_hat = []$ for i in range(self.k): row = []for j in range(self.k): row.append(SampleNTT(XOF(rho, i, j), self.q)) a\_hat.append(row) e1 = [SamplePolyCBC(PRF(r, i+self.k, self.n2), self.q, self.n2) for i in range(self.k)] e2 = SamplePolyCBC(PRF(r, 2\*self.k, self.n2), self.q, self.n2) u = [] for i in range(self.k): sum = e1[i]for j in range(self.k): # transposta de A\_hat por trocar o i com o j u.append(sum)  $\mu = Decompress(ByteDecode(msg, 1), 1, self.q)$ v = [0]\*256for i in range(self.k):  $v = polyADD(NTT_inv(NTT_mult(t_hat[i], r_hat[i], self.q), self.q), v, self.q)$  $v = polyADD(v, polyADD(e2, \mu, self.q), self.q)$ 

return c1 + c2

W = [0]\*256for i in range(self.k): w = polySUB(v, w, self.q)m = ByteEncode(Compress(w, 1, self.q),1) return m Test K PKE In [ ]: test1 = K\_PKE(256, 3329, 2, 3, 2, 10, 4) ek\_PKE, dk\_PKE = test1.keygen() print("Encryption Key (ek\_PKE):", ek\_PKE) print("Tamanho da Encryption Key: ", len(ek\_PKE)) print("Decryption Key (dk\_PKE):", dk\_PKE) print("Tamanho da Decryption Key: ", len(dk\_PKE)) msg = b"Esta e uma mensagem de 32 bytes!" print("Tamanho da mensagem: ",len(msg)) r = Random\_32\_bytes() cyphertext = test1.PKE\_Encrypt(ek\_PKE, msg, r) decrypt\_msg = test1.PKE\_Decrypt(dk\_PKE, cyphertext) print("Decrypt message: ", decrypt\_msg) assert(decrypt\_msg == msg) Tamanho da Encryption Key: 800

Tamanho da Decryption Key: 768 Tamanho da mensagem: 32 Decrypt message: b'Esta e uma mensagem de 32 bytes!' ML-KEM Key-Encapsulation Mechanism class KEM: def \_\_init\_\_(self, kem\_type): self.n = 256self.q = 3329self.sharedkeysize = 32 self.type = kem\_type **if** kem\_type == 512: self.k = 2self.n1 = 3self.n2 = 2self.du = 10self.dv = 4self.enckeysize = 800 self.deckeysize = 1632 self.ciphersize = 768 **if** kem\_type == 768: self.k = 3self.n1 = 2self.n2 = 2self.du = 10self.dv = 4

self.enckeysize = 1184 self.deckeysize = 2400 self.ciphersize = 1088 **if** kem\_type **==** 1024: self.k = 4self.n1 = 2self.n2 = 2self.du = 11self.dv = 5self.enckeysize = 1568 self.deckeysize = 3168 self.ciphersize = 1568 self.PKE = K\_PKE(self.n, self.q, self.k, self.n1, self.n2, self.du, self.dv) def KeyGen(self): z = Random\_32\_bytes() ek\_pke, dk\_pke = self.PKE.keygen() ek = ek\_pke  $dk = dk_pke + ek + H(ek) + z$ # Verificar parametros assert len(ek) == self.enckeysize assert len(dk) == self.deckeysize return ek, dk def Encaps(self, ek): msg = b"Shared secrect key com 32 bytes!" print("Chave partilhada: ", msg) print("Tamanho da mensagem: ",len(msg)) ghash = G(msg + H(ek))k = ghash[:32]r = ghash[32:]c = self.PKE.PKE\_Encrypt(ek, msg, r) # Verificar parametros assert len(k) == self.sharedkeysize assert len(c) == self.ciphersize return k, c def Decaps(self, c, dk):  $dk_pke = dk[:384*self.k]$  $ek_pke = dk[384*self.k : 768*self.k + 32]$ h = dk[768\*self.k+ 32 : 768\*self.k + 64]z = dk[768\*self.k + 64 : 768\*self.k + 96]mdash = self.PKE.PKE\_Decrypt(dk\_pke, c) ghash = G(mdash + h)kdash = ghash[:32]rdash = ghash[32:]

kbar = J(z + c)

if cdash != c: **return** kbar

return kdash

test ML-KEM-512

ek\_KEM, dk\_KEM = kem1.KeyGen()

Tamanho da mensagem: 32

Tamanho da Shared Key: 32

Tamanho da cifra: 768

Tamanho da cifra: 1088

Tamanho da Shared Key: 32

ML-KEM key-Encapsulation Mechanism Valid!!

test ML-KEM-1024

Tamanho da Encryption Key: 1568

Tamanho da cifra: 1568

Tamanho da Shared Key: 32

ML-KEM key-Encapsulation Mechanism Valid!!

Tamanho da Shared Key: 32

print("Encryption Key (ek\_KEM):", ek\_KEM)

print("Decryption Key (dk\_KEM):", dk\_KEM)

shared\_key1, cyphertext = kem1.Encaps(ek\_KEM)

shared\_key2 = kem1.Decaps(cyphertext, dk\_KEM)

In [ ]: kem1 = KEM(512)

print("Shared key: {}\nTamanho da Shared Key: {}".format(shared\_key1, len(shared\_key1))) if shared\_key1 == shared\_key2: print("ML-KEM key-Encapsulation Mechanism Valid!!") else: print("ML-KEM key-Encapsulation Mechanism Invalid!!") Tamanho da Encryption Key: 800  $x16F\xdd\xf3\x0782\xd5\x1e\xe8\xf9U\xe8\xe7tn'$ Tamanho da Decryption Key: 1632 Chave partilhada: b'Shared secrect key com 32 bytes!'

ML-KEM key-Encapsulation Mechanism Valid!! test ML-KEM-768 In []: kem1 = KEM(768)ek\_KEM, dk\_KEM = kem1.KeyGen() print("Encryption Key (ek\_KEM):", ek\_KEM) print("Tamanho da Encryption Key: ", len(ek\_KEM)) print("Decryption Key (dk\_KEM):", dk\_KEM) print("Tamanho da Decryption Key: ", len(dk\_KEM)) shared\_key1, cyphertext = kem1.Encaps(ek\_KEM) print("Shared key: {}\nTamanho da Shared Key: {}".format(shared\_key1, len(shared\_key1))) print("CypherText: {}\nTamanho da cifra: {}".format(cyphertext, len(cyphertext))) shared\_key2 = kem1.Decaps(cyphertext, dk\_KEM) print("Shared key: {}\nTamanho da Shared Key: {}".format(shared\_key1, len(shared\_key1))) if shared\_key1 == shared\_key2: print("ML-KEM key-Encapsulation Mechanism Valid!!") print("ML-KEM key-Encapsulation Mechanism Invalid!!")

\x15\xbf\'\xc6R\xa1\xa9\xa5\xb1\x11\x91\xbd\xed;E\xe6\xb0\x03\xe1\x94\x1f\x89;\xferH' Tamanho da Encryption Key: 1184 Tamanho da Decryption Key: 2400 Chave partilhada: b'Shared secrect key com 32 bytes!' Tamanho da mensagem: 32 Shared key:  $b"\x86\n\x6\xf9kM\xd3\x0c\x13n\xd2'x\x9dw^\xed\xcet>\x93\xd3mCV}P\xcbH\xe9\x05\x90"$ Tamanho da Shared Key: 32

Shared key: b"\x86\n\xa6\xf9kM\xd3\x0c\x13n\xd2'x\x9dw~\xed\xcet>\x93\xd3mCV}P\xcbH\xe9\x05\x90"

In [ ]: kem1 = KEM(1024)ek\_KEM, dk\_KEM = kem1.KeyGen() print("Encryption Key (ek\_KEM):", ek\_KEM) print("Tamanho da Encryption Key: ", len(ek\_KEM)) print("Decryption Key (dk\_KEM):", dk\_KEM) print("Tamanho da Decryption Key: ", len(dk\_KEM)) shared\_key1, cyphertext = kem1.Encaps(ek\_KEM) print("Shared key: {}\nTamanho da Shared Key: {}".format(shared\_key1, len(shared\_key1))) print("CypherText: {}\nTamanho da cifra: {}".format(cyphertext, len(cyphertext))) shared\_key2 = kem1.Decaps(cyphertext, dk\_KEM) print("Shared key: {}\nTamanho da Shared Key: {}".format(shared\_key1, len(shared\_key1))) if shared\_key1 == shared\_key2: print("ML-KEM key-Encapsulation Mechanism Valid!!") print("ML-KEM key-Encapsulation Mechanism Invalid!!")  $f \times 90 = \int_{x_0} f^{x_0} f^{$ 

Tamanho da Decryption Key: 3168 Chave partilhada: b'Shared secrect key com 32 bytes!' Tamanho da mensagem: 32 Shared key:  $b'\times11\times fe\times ac\times 4\times83\times92 f^\times f0\times11-xd0X/xfb\times1d\times ac\times b^*u\times95\times13\times ac^*xc4\times91\times g^*xf0\times g^*u$ 

 $$$ x01\times0eQ\times01\times15\timescc\timesn\times1+xa3\times9\timese0\times14S\timesc8)E\timesf1\timesa8\times03$ 

Shared key:  $b'\times11\times fe\times xac\times 4\times83\times92 f^\times f0\times11-xd0X/xfb\times1d\times xac\times xba!w\times95\times13\times xc4\times y91\times y61\times xaf\times xba.$ 

y = sum(bits[2\*i\*eta+eta+j] for j in range(eta))

 $f_{out}[j + length] = (zeta * (f_{out}[j + length] - t)) % Q$ 

row.append(SampleNTT(XOF(rho, i, j), self.q)) s\_hat = [NTT(SamplePolyCBC(PRF(sigma, i, self.n1), self.q, self.n1), self.q) for i in range(self.k)] e\_hat = [NTT(SamplePolyCBC(PRF(sigma, i+self.k, self.n1), self.q, self.n1), self.q) **for** i **in** range(self.k)] sum = polyADD(NTT\_mult(a\_hat[j][i], s\_hat[j], self.q), sum, self.q) r\_hat = [NTT(SamplePolyCBC(PRF(r, i, self.n1), self.q, self.n1), self.q) for i in range(self.k)] sum = polyADD(NTT\_inv(NTT\_mult(a\_hat[i][j], r\_hat[j], self.q), self.q), sum, self.q)

c1 = b"".join(ByteEncode(Compress(u[i], self.du, self.q), self.du) for i in range(self.k)) c2 = ByteEncode(Compress(v, self.dv, self.q), self.dv) def PKE\_Decrypt(self, dk\_pke, cypher): c1 = cypher[:32\*self.du\*self.k] c2 = cypher[32\*self.du\*self.k:] u = [Decompress(ByteDecode(c1[i\*32\*self.du:(i+1)\*32\*self.du], self.du), self.du, self.q) for i in range(self.k)] v = Decompress(ByteDecode(c2, self.dv), self.dv, self.q)  $s_hat = [ByteDecode(dk_pke[i*384:(i+1)*384], 12)$  for i in range(self.k)] $w = polyADD(NTT_inv(NTT_mult(s_hat[i], NTT(u[i], self.q), self.q), self.q), w, self.q)$  $$ x98d1{xa1Wx17^xd582\t\timesba\x0b\xfb\xabYKs\x86d\x02}K\xd8W\xddY0\xef\x10\xa8\xa4F\xfe\xd0\xbfx\xde\xff\x04\x17\x9b\xe5o'$ 

 $bc \x 96 \x 10 \x b6 \x e3 \nu \x 08 \x 93 \x 14Z \x fbw \x b6 \x e3 \x e0 \$  $1a\xb6\xc3\x92 \xbe\xc3\x88C)6\xcdE\x7f\x9b\x98\x13\x86\xaf\x89\x9981i\x19y\xe4\&\xf7\xa0\xbf\x91\xf9\x13\x86\x9f\xe2@{\xcc[\'0\x87\xa3\xcc0\x17F\x9f\x9f\x9f\xe2@{\xcc[\'0\x87\xa3\xcc0\x17F\x9f\x9f\xe2@{\xcc[\'0\x87\xa3\xcc0\x17F\x9f\x9f\xe2@{\xcc[\'0\x87\xa3\xcc0\x17F\x9f\x9f\xe2@{\xcc[\'0\x87\xa3\xcc0\x17F\x9f\x9f\xe2@{\xcc[\'0\x87\xa3\xcc0\x17F\x9f\x9f\xe2]}}$ \xd3yN\x8bG\x91\xf5\xcd<H(0\x0by\xe6\x899\x1cY0x\x16nw4dw\xd9H\r\xc2\xc0\x08\xf2\xb1\x01du\xc8\xae\xe2c\xa0\xb8\x1b\x94\xf0F\x9f\xd97v\x15~\x02\xa8>\xe0@6\x8c\x07\tp\xc5\xb2\xeb\xc7\x10'  $51 \% \times 51 \% \times$  $\x 17\x 12\x 04\\Rx\x fa\{x f4\x 07\x c9\x 0a\x 11\x 83\x c9\x 9a\x 10\x fe\x 10\x c4\x f2\x c9\x 11\x 9f5\x b6TI\x d0'\}$ 

cdash = self.PKE.PKE\_Encrypt(ek\_pke, mdash, rdash) assert len(kdash) == self.sharedkeysize

print("Tamanho da Encryption Key: ", len(ek\_KEM)) print("Tamanho da Decryption Key: ", len(dk\_KEM)) print("Shared key: {}\nTamanho da Shared Key: {}".format(shared\_key1, len(shared\_key1))) print("CypherText: {}\nTamanho da cifra: {}".format(cyphertext, len(cyphertext)))  $Encryption Key (ek_KEM): b'cf4\x8e\xf9`26\x18\'\x04!\x10\x15\\x04!\x03\x15\x10\x15\x17\xc4\x08\x42\x07\xa4\x07\xa4\x05\x10\x15\x17\xc4\x08\x42\x10\x15\x17\xc4\x08\x42\x10\x15\x17\xc4\x08\x42\x10\x15\x17\xc4\x08\x42\x10\x15\x17\xc4\x08\x42\x10\x15\x17\xc4\x08\x42\x10\x15\x17\xc4\x08\x42\x10\x15\x17\xc4\x08\x42\x10\x10\x15\x17\x15\x17\x$ \xad|J<g\x17\xc9vQ\xf3\xc0\x0b\x06\xa1K\x8c\xc8\\n\x15\x0b\x06\\x15\x02\x8e\xe5o\x1e9\xae~\xbb\x8a8Kd\xf9\xc2\x8e\xe5o\x1e9\xae\xb7\\\'\xeb\xc7\xa6.\xe3\x90\) \xe4e\xc8-|~MUT\xc3!%t\x90\xc2\x92ppI\xac\x90\xc9\xe6p\xa7qF\xf34R\x1e\xa4\xad\x08\x08\x08\x08\x931\xae\x86e\x19\x17\xf5\x17\x12\x81\x88\x92\xe0p\xa7qF\xf34R\x1e\xa4\xad\x08\x08\x08\x08\x18\x93f.\x9c  $Decryption Key (dk_KEM): b'oq$\x18\xc2\xab\xee\xea=\n\x83\x9fIZ7n\x82\xb7\xe6\xf18\tq^{xb1\xee\xea}\x05\x90\x1f\'\x96Z\nq\xb6\xe6\xf18\tq^{xb1\xee\xea}\x01\x1b\x1f\xee\xea}$ 

Shared key: b'v\x1d4~\*\xc8Xf\xea5\x0cN\xfei\xfc>\xd6{\xe3w\xc6\x1f\xb4\xbe;1\xcezJ\xbb6S' 0, Hc\x7f\xd9p\xb1\xe3\x86\x8f\xe0"\xff[c\x04\xc7\x83\xcd\xc8,\xca=\xb5\xc2[v\x91w)\xc6o\xffz#,\x8a\xcb(\xf7\x12\xda\xb6\x060Jbq\xeb\x91:\xa8\xbbdr\x97e\xb9' Shared key:  $b'v\x1d4^*\xc8Xf\xea5\x0cN\xfei\xfc>\xd6{\xe3w\xc6\x1f\xb4\xbe;}1\xcezJ\xbb6S'$ 

Uee\x8a9\xb4rJI\x91\xcc\xa6\n\xf1\x03\xd9\x90p\x94\x94\x7fu)\xb0I+^DD\xcb\x4x94\x7fu)\xb0I+^DD\xcb\x4x96\xb9f\xb7\xee\xe5\x82\*\xf2\x8f\x8e\x14\x1f\xf8t\xac\* \xc1\x11\xc3\x7f\'h\xc0Z\x15B\x84y\xcf&\x16Q\nYoe\xe2\xa b/\xfc.\xef\xea\x96\x9a\x96\x9a\x9cw.Eoj\xa7\xc39\x80T\x92C\xc1\xf5\x12v\xe8[!\x161\x0c\xf9\xa0bE1U\xa1\xb7\x83\x86\xab\xe1\x97\x10\x15f\x90Z\xccMz\xx8f\x93\x a}\x91\x8e\xa3rq\x91\xe3/H\xf6\x17\x18S\x1f\x8a\xa07\xb0\\x88\x83\\xfa\xa08\xfa\xa08\xfa\xa08\x15~e\xb0DD\xc5\x86\xe2\x8b2\xd4\x1b2\x0b\xe5\x99Z(2P\*\x10\\\x8b\xa7\xc8\x80\x15\xd0\\x8b\xa7\xceD\xd6;\xdcA 9!\xa4\\xb5d<\x15\\xc3\\x9efM\xf5\'p\\xcaC\\xc9\\x05\\x8d\\x4\\xa3\\x4\\x10\\x91\\x11\\x19\\xf8\\xa4\\x10\\x91\\x11\\x19\\xf8\\xa3\\x4\\x33\\x4\\x33\\x95\\x8d\\x4\\x33\\x33\\x  $5\x88\x85\x99$ , \xb8\xe67\x0c\xfci\x17\x1bd\n\xbf\xe4Q\x1d\xaf\\x90\x8f1\x98\xb0\x90\x8f1\x98\xb0\x90\x8f1\x98\xb0\x90\x8f1\x98\xb0\x90\x8f1\x98\xb0\x90\x8f1\x98\xb0\x90\x8f1\x98\xb0\x90\x8f1\x98\xb0\x90\x8f1\x98\xb0\x90\x8f1\x98\xb0\x90\x8f1\x98\xb0\x90\x8f1\x98\xb0\x90\x8f1\x98\xb0\x90\x8f1\x98\xb0\x90\x8f1\x98\xb0\x90\x8f1\x98\xb0\x90\x8f1\x98\xb0\x90\x8f1\x98\xb0\x90\x8f1\x8b\x90\x8f1\x98\xb0\x90\x8f1\x98\xb0\x90\x8f1\x98\xb0\x90\x8f1\x8b\x90\x8f1\x98\xb0\x90\x8f1\x98\xb0\x90\x8f1\x98\xb0\x90\x8f1\x8b\x90\x8b\x90\x8f1\x8b\x90\x8b\x8b\x90\x8b\x90\x8b\x90\x8b\x90\x8b\x90\x8b\x90\x8b\x90\x8b\x90\x8b\x90\x8b\  $\x 001 \times 91 \times 901 \times 90$ 5\xab\x85\x80\\\xb0\xb0\xb6\xe1\x9a\xe1\x9a\xe1\x9a\xe1\x9a\xe1\x9a\xe1\x9a\xe1\x9a\xe1\x9a\xe1\x9a\xe1\x9a\xe5\x40\xb5\xxb5x\xdb\xa2\xab\x17\xf3\x93o\xfczr\x11\x014\\x6\x03\x9\x6\x9x\x45\x80\\\x18\xa9\xa6\x03\x9\x17\xf3\x93o\xfczr\x11\x014\\x6\x03\x0  $c5\x14\%\xd8D\xc7\xd2\xd2\x80\xc2\x12\x96R\xd9\x84\x97\x85\xc3\x8c\xd1eT\x83\xe5\x10(\x96R^*5\x$ ^DD\xcb\xdc\xcc\xcd\xc5D\x98\x89\x89\x89\x89\x89\x89\x69\x95T\x14-m\x9c\x84k8yB\xc0\xbf\x95T\x14-m\x9c\x89\x60\xbf\x9c\x6\x15\x12v\xe  $4\x1f\x89;\xferH\x04\x90\xac\xc6\xa4\x90\x01\xe2\xa8\x1cP\x05\xab\xbf\xa9A\xf2->H\xd3\xbf8+\x84\x0f\x13\x14\xea\xac\xc4^=\xe0\xeb7!\xf4\xf5\x12[#?e\x826\xe9\xac\)\xef\xf0\xc6\xf6\xf6\xf6\xf6\xf6\xfb\xcf\x1fgb;\xa8\xe7'$ 

\x17\xf3@\x1e\xc1eh5%\xcf\x8bH\xcdP4\x00\xd7\x83f\x16I\xc0p\xa7\x88\xd4\xa7\[5\r\xcb\xb7\xaf\xe1U\x0b\x95\x06\xe8\xf3\x84\\xd5\xb6\x95\x05\xd7\xx8\xd5\xx8\xd5\xx8\xd5\xx8\xx85\xx87\xxb0JK\xd5\xb6\x95\xx05\xx8\xx45\xx8\xx85\xx87\xxb0JK\xd5\xxb0JK\xd5\xxb0JK\xd5\xxb0JK\xd5\xxb0JK\xd5\xxb0JK\xd5\xxb0JK\xd5\xxb0JK\xd5\xxb0JK\xd5\xxb0JK\xd5\xxb0JK\xd5\xxb0JK\xx  $2\x04H\x1d\x9a\x96jSE\xd6P\x18\xf0\x8c\xcc\x99@\x1bi9\x1ac\xfe\xbdQD\x13\xef*\xd6\x9e]L[\n\xeepwV\xec\xefX/M\xfeZ\xaer\xe5\xd8']L[\n\xeepwV\xec\xefX/M\xfeZ\xaer\xe5\xd8']L[\n\xeepwV\xec\xefX/M\xfeZ\xaer\xe5\xd8']L[\n\xeepwV\xec\xefX/M\xfeZ\xaer\xe5\xd8']L[\n\xeepwV\xec\xefX/M\xfeZ\xaer\xe5\xd8']L[\n\xeepwV\xec\xefX/M\xfeZ\xaer\xe5\xd8']L[\n\xeepwV\xec\xefX/M\xfeZ\xaer\xe5\xd8']L[\n\xeepwV\xec\xefX/M\xfeZ\xaer\xe5\xd8']L[\n\xeepwV\xec\xefX/M\xfeZ\xaer\xe5\xd8']L[\n\xeepwV\xec\xefX/M\xfeZ\xaer\xe5\xd8']L[\n\xeepwV\xec\xefX/M\xfeZ\xaer\xe5\xd8']L[\n\xeepwV\xee\xefX/M\xeepwV\xee\xefX/M\xeepwV\xee\xefX/M\xeepwV\xe$ 

\xbd\xdee\x08\xf1\x93`o\xdcH\xc2\x88\x06\xb3\xf9\x95\xd8\xb7\xc9\x98\x06\x08\xxf1\x93\x6\x08\x6\x94\x80\x99\x02\x83\xbd\xc5\x94\x80\x02\x83\xbd\xc5\x94\x80\x06\x08\x06\x08\x06\x08\x02\x83\xbf\xc5\x94\x80\x02\xbf\xc5\x94\x80\x06\x08\x08\x02\xbf\xc5\x94\x80\x02\xbf\xc5\x94\x80\x02\xbf\xc5\x94\x80\x06\x08\x02\xbf\xc5\x94\x80\x02\xbf\x02\xbf\x02\xbf\x02\xbf\x02\xbf\x02\xbf\x02\xbf\x02\xbf\x02\xbf\x02\xbf\x02\xbf\x02\xbf\x08\xbf\x02\xb  $4q \times d_x = \frac{4q \times d_x \times d_x$ \xf80RcM\xdf\xb8p\x92\x80\$"P\x0f\xe0\x82\x9e\xa0\x81\x9d\xf2\x19H\x03\xa1\x9x\x81\x94\xef\xa6\x\x61\x94\xef\xa6\x\x82\x80P\x01\x\03\x\xa1\x\03\x\03\x 8, v x f b x f c x 81 x 026 V x A x 9 b x 26 V x A x 9 b x 26 V x A x 9 b x 26 V x 14 x 27 f x 69 V x 26 V x 14 x 27 f x 69 V x 26 V x 14 x 27 f x 69 V x 26 V x 14 V x 27 f x 69 V x 26 V x 14 V x 27 f x 69 V x 26 V x 14 V x 26 V x $f7 \times f^{xaf} \times$  $X/M \times feZ \times aer \times e5 \times d8A \times bc \times e6 \times e14 \times e19 \times e$ 

\xea?\xd2\x08\\xd4\x98\\x0f\\xac\x12KS\x829S\\x9a5\\x15\\x05S\\x03\\xb6\\xb7\\xaa\\xa6\\xfd\\x90\\x90\\x90\\x05\\x03\\xb6\\xb7\\xaa\\xa6\\xfd\\xe9\\x\fd\  $e6\xd00\xb71\x00\xd4\&\x80\x9a5\x05\xb4\x85\xc6\xb6\xb2\xb3\xa1\x975\x9a\xfe\xa1\x975\x9a\xfe\xa5\xb2\xb3\xb2\xb3\xc5\xc4]$ \xbe+\x8f\x0c\xb9mo\x040\xad\*\xa7\x0b\x80\xc6\xe1\xa7KE\xe1\x9d\x15\x18\xb1G<\xce\x8f\x0x\x90\xaf\xe2\xf8=\xbee&\xabX\xae\x6\xcb\x89FX\\\xae\x89FX\\\xae\x89FX\\x89FX\\x89FX\\\xae\x89FX\\x89FX\\x89FX\\x89FX\\x89FX\\\x89FX\\x89FX\\x89FX\\\x89FX\\\x89FX\\\x89FX\\\x89FX\\x89FX\\x89FX\\\x89FX\\x89FX\\x89FX\\\x89

 $\x d = x d$ E\x8a\xa3\\x8a5\x93\x96\t\xb6\xa0\x9a\x9b\x9e\p\x9e\p\x9e\p\x81\x82\xb6\x90\x9e\p\x8a\x82\xb6\x9o\x9e\p\x8a\x81\x81\x8b\xca\r\x17#\x103\x1c\xc7\xf4&\x121\x8b\xca\r\x17\xc3e.\x81\x8b\xca\r\x15\x82\xb6\x9o\x9e\p\  $3u\x90\x08\x06\xc0\x1bp\x08kR\xce)D\xbf21cf4\x8e\xf7\xc4\xbf[\x02\x86\xf1\x03\x1c\x06)^x$  $5\x174\p\x92\x88\x85\xeal)\x8c\x13\x11\x83K\xbc\xed\x90\xea\x91\x86\x10\x06\x10\x86\x10\xe0\xe1.\L\x13N\x0bB\xe1.\L\x13N\x0bB\xe1.\L\x13N\x$ b\n\\\xb4\x19\xad|J<g\x17\xc9vQ\xf3\xc0\x0b\x06\xa1K\x8c\xc8\\n\x15\xb0\x06\x\1e9\xae\x5\xbb\\x8a\xc0\x0b\x8a\xc0\x\06\x\1e9\xae\x5\x\bb\\x8a\xc0\x\06\x\1e9\xae\x5\x\bb\\x8a\xc0\x\06\x\1e9\xae\x5\x\bb\\x8a\xc0\x\06\x\1e9\x\ae\x5\x\bb\\x8a\xc0\x\06\x\1e9\x\ae\x5\x\bb\\x8a\xc0\x\06\x\1e9\x\ae\x5\x\bb\\x8a\x\c0\x\06\x\1e9\x\ae\x5\x\bb\\x8a\x\c1e\x\b1\x\7fq\x\cep\x\ab\x\1e9\x\ae\x\bf\\\'\\\\'\\x\eb\x\c1e\x\bf\\x\for\x\for\x\for\x\for\x\for\x\for\\\\'\\x\eb\x\for \x08\x01\xda\xdb\xb7\xf6\xc3\x99\x88g\xac4\\xbe\xa6\x1a\xae\x12\x8bY\x9e\xbf\\\nvF\xbe\x4j\x90\xe5\x60\x13\xx61\x01\r0\xe5\xbo\x4j\x90\x65\x60\x13\xx61\r0\x83\xx61\x01\r0\x83\xx61\x\da\x4\xbb\x45\x\da\x4\xbb\x45\x\da\x4\x\bb\x45\x\da\x45  $11f\x0c\xddn\x06b\xb2z\xf8_0N_\xbf\xd7\xg0\xbf\xd7\xg0\xbf\xd7\xg0\xbf\xd7\xg0\xbf\x$ 18\xb2\xac[\x96=\xe7\x19+>\xc6\xf8\xed!I\xba\xf5\xaa\x05\xe28R\\\\\xb1\x96\xf6\xfb\x9f\x8b\xf0\x1f\xbc\x01v\xf  $Encryption Key (ek_KEM): b'\x80\xb3\xab\xe6\xa1T\x01V\x0e^vu\x81\xf7\x86\$) \times (ek_KEM): b'\x80\xb3\x85\x0f\x02g\xf0\x4^{x05}\x87\x02g\xf0\x4^{x05}\x87\x02g\xf0\x87\x02g\x67\x02g\xf0\x87\x02g\xf0\x87\x02g\xf0\x87\x02g\xf0\x87\x02g\xf0\x87\x02g\xf0\x87\x02g\xf0\x87\x02g\xf0\x87\x02g\xf0\x87\x02g\xf0\x87\x02g\xf0\x87\x02g\xf0\x87\x02g\x67\x$  $7 \left( x_{3} x_{5} x_{4} x_{2} x_{9} x_{1} x_{2} x_{9} x_{1} x_{2} x_{2} x_{1} x_{2} x_{2}$ dd\xe5c\xbb\xf3R\xcc5C\x13\xf5\xad6\x19\x9d\x1b\xab\xa6\x0c\x05\x9e\x89\x84\xa7\x10|\x08V\x86\xac\r\xf1vR\x10T\xbb\x89\x84\xa7\x10|\x08V\x86\x0c\x05\x9e\x85\x96\x08\x89\x89\x89\x89\x80\x10\x0c\x6\xb0\x19\xf7\x8 a"\xc5\xb4\x8a\x84,\xce\xe5\x07)\xb8\x91\x0c\xc1\x96\x8b\x07\x00\x13\x19\x0c\xc1\x96\x85\x08\x001\\\\x07\x20\\x6\x85\x08\x98\x13\xe4\x82\xa4\x81\x86\x85\x08\\x001\\\\x07\x20\\x13\x19\\x\\x01\x96\x85\x01\\\x01\x96\x85\x98\x13\x\\x01\\x01\\x01\\\x 

\' J\xa7&(\xf3\xa6\x9a\xadI\xa3\xf8\xaa\x1f!|\xbe\xc5@7A\xec!\xd5\xf5\x015\_\x95\xf0fb\xf0y\xe9\xe2\x9d\xda\x1c\xcf\xf2\xf1/\x \\\x\d1\\x83\xe2\xbdRE\x18\xba,\xdd\xadI\xa3\xf8\xaa\x1f!\xbe\xc5\x95\xf0fb\xf0y\xe9\xe7\x1c\x\da\xxf1\x\x\f1\x\f1\x\x\f1\x\f1\x\f1\x\x\f1  $81\xbf\xf5\x93N\xf0t\\xba\x95\xba\x95\xba\x80\xa1\x80$  $b2\xbe\xa4\x88\xd9?\xb8\x1c\x8b\x1c\$ 

4\xd4(SN\nNrx|\xe0UZ\x8a\xc1\xc2\\\xf9\\xf9\xxf9\xx66\xb3(\xf8\x8f<:u%\x9a\x18\x8c\xe0|n\x04E\'+eb\x93Uy\xb40\xb5\x85\x93\x06L\xa8N\x11\xa6\x89\x18\x8c\xe7\xa6\xb7\x13P\*:\x  $ea) \ x00 \ x01 \ x01$ \x99\xa6M!\\$\x92\x83\xd73\xd0\x00\x09\\x90\\xa6\x2\\xc1\xee\xa1\x97\\\xa3\x7f\xd8\r\x18\x7f\x48\r\x18\x7f\x18\x7f\x48\r\x18\x7f\x48\r\x18\x7f\x48\r\x18\x7f\x48\r\x18\x7f\x s:r\xe0\x83\x04\xb8\x19AG\xa9\xbfn\x90v\xdc\xaa\x16J\x80Ae\x95\_T\xd1\xc4\xd5\x89\*\xb0V{\x02|\xaa\x20\x87\xx6\x05\x07\xc2\xxa1\x04\x85\x14b\xa7\x83\x90r\x83\x90r\x83\x90r\x83\x90r\x83\x90r\x83\x90r\x83\x90r\x83\x90r\x83\x90r\x83\x90r\x83\x90r\x83\x90r\x85\x10\xbdV\xb3n\x  $CypherText: b'\tg:5\x08Abk\xd0\x08\xc4\x10\x060\xb7\xa4\xc8p(\xa6\xd41\xc3C9\xd6\xd4)\x60\xb7\xa4\xc8p(\xa6\xd4)\x60\xb7\xa4\xc8p(\xa6\xd4)\x60\xb7\xa6\xd4)\x60\xb7\xa6\xd4)\x60\xb7\xa6\xb1\xb60\xb7\xa6\xb1\xb60\xb7\xa6\xb1\xb60\xb7\xa6\xb1\xb60\xb7\xa6\xb7\xa6\xb7\xa6\xb7\xa6\xb7\xa6\xb7\xa6\xb7\xa6\xb7\xa6\xb7\xb6\xb7\xa6\xa6\xa6\xa6\xa6\xa6\x$ bb x c6 x cb x e5 L x 13 x a 97 x d0 x c9 J x 1 f b x d8 x cd x f f M x b 6 T x 8a x f 0 n 6v : x da x x 94 x c 9 x 2 T x da x 2 J x e c x 8d x a 4 x 11 x 9a x f a x a 0 x c 9 J x 1 f b x d 8 x x f 0 n 6v : x d a x x d a x x f 0 n 6v : x d a x x d a x x d a x $I \ xaa \ x07V \ x18f \ x05V \ x11 \ x05V \ x6V \ x01V \ x08V \ x01V \ x05V \ x05V$  $1 \times 9d'' \times 0.05 \times 1000 \times 100$  $z\x 0\x 91B/\x 00\x 0x 15\x 0x 0x 15\x 0x$  $f1\xc0c^f\xd8\xc8YM\xff\xcc\xd4nU\xf2\x03\xa4\xe8\x85(\xeb\x92\xbd\x90\x1aY\x94D\xf2\xf4\xbc\x91\xc9\xd6\x92\xbd\x90\xa6\x92\xbd\x92$