Enunciado

3. Usando a experiência obtida na resolução dos problemas 1 e 2, e usando, ao invés do grupo abeliano multiplicativo $\mathbb{F}_p^*$, o grupo abeliano aditivo que usou na pergunta 2,
   A. Construa ambas as versões IND-CPA segura e IND-CCA segura do esquema de cifra ElGamal em curvas elípticas.
   B. Construa uma implementação em curvas elípticas de um protocolo autenticado de "Oblivious Transfer" κ-out-of-n.

Exercício 3.a

Versão IND-CPA

Geração de chaves

Chave Privada (sk): Um número inteiro aleatório no intervalo $[1, L-1]$, onde L é a ordem do subgrupo gerado pelo ponto base P.
Chave Pública (pk): Um ponto na curva elíptica, calculado como $p_k = s_k * P$

Cifra

- Escolher um número inteiro aleatório κ no intervalo $[1, L-1]$.
- Calcular o ponto $C_1 = \kappa * P$.
- Converter a mensagem m num ponto na curva elíptica (como no método de Koblitz).
- Calcular o ponto $C_2 = m + \kappa * p_k$.
- O texto cifrado é o par (C_1, C_2)

Decifra

- Usar a chave privada s_k para calcular $m = C_2 - s_k * C_1$.
- Converter o ponto m de volta para a mensagem original

Diferenças entre usa o grupo multiplicativo F_p^* e usar um grupo abeliano

No Exercício 3.A, a cifra e decifra não são iguais ao Exercício 1.A, porque as operações matemáticas mudam:

- No grupo multiplicativo F_p^*, usamos exponenciação e multiplicação modular.
- Nas curvas elípticas, usamos adição de pontos.

Portanto, a cifra e decifra no Exercício 3.A devem ser adaptadas para as operações no grupo aditivo da curva elíptica. Comparando-as:

Operação	**Grupo Multiplicativo $\mathbb{F}_p^*$**	**Curva Elíptica E**

Operação	Grupo Multiplicativo $\mathbb{F}_p^*$	Curva Elíptica E
Geração de Chaves (pk)	$g^s \mod p$	$g \cdot s$
Cifra	$\gamma = g^\omega \mod p$	$\gamma = \omega \cdot g$
	$\kappa = (g^s)^\omega \mod p$	$\kappa = \omega \cdot S$
	$\delta = m \cdot \kappa \mod p$	$C = M + \kappa$
Decifra	$\kappa = \gamma^s \mod p$	$\kappa = s \cdot \gamma$
	$m = \delta \cdot \kappa^{-1} \mod p$	$M = C - \kappa$

fonte: https://crypto.stackexchange.com/questions/9987/elgamal-with-elliptic-curves

Conversão de pontos

Especialmente no contexto de curvas elípticas, a conversão de pontos é uma técnica utilizada para mapear mensagens (ou dados) em pontos específicos sobre a curva elíptica. Para conseguir agregar o melhor dos dois mundos (curvas elípticas e ElGamal) precisamos de mapear mensagens em pontos e vice-versa.

Nesta secção apresentamos a metodologia de conversão que iremos utilizar:

Para mapear uma mensagem de $p - 1 - l$ bits num ponto (x, y) sobre uma curva elíptica definida no corpo primo abeliano de p bits fazemos passo a passo:

1. **Preparação da Mensagem:**
   - A mensagem m tem $p - 1 - l$ bits.
   - Para ajustar o tamanho, a mensagem é concatenada com l bits zero, resultando em $x = m \parallel 0l$.
2. **Cálculo de x':**
   - O valor x' é calculado como $x' = x^3 + a \cdot x + b \mod p$, onde a e b são parâmetros da curva elíptica, e p é o primo que define o corpo finito.
3. **Verificação do Resíduo Quadrático**:
   - Se x' for um resíduo quadrático módulo p, então existe um y tal que $y^2 \equiv x' \mod p$. Nesse caso, o ponto (x, y) é um ponto válido na curva.
   - Se x' não for um resíduo quadrático, os últimos l bits de x são incrementados em 1, e o processo é repetido.
4. **Limite de Tentativas:**
   - O processo é repetido até 2^l vezes. Se nenhum x' válido for encontrado, a mensagem é considerada "não codificável".
5. **Decodificação:**
   - Para recuperar a mensagem original, basta ignorar a coordenada y e remover os últimos l bits da coordenada x.

fonte: https://crypto.stackexchange.com/questions/76340/how-to-create-an-ec-point-from-a-plaintext-message-for-encryption

Implementação

Usando a classe definida no exercício 2:

In [1]:
```
import hashlib
import random
from sage.all import *

class EcDSA_Ed25519:
    def __init__(self, p, a, d):
        assert a != d and is_prime(p) and p > 3
        self.K = GF(p)  # Definido como atributo para possível uso futuro

        # Convertendo a e d para elementos do campo finito K
        self.a = self.K(a)
        self.d = self.K(d)

        # Calculando A e B dentro do campo finito K
        A = 2 * (self.a + self.d) / (self.a - self.d)
        B = 4 / (self.a - self.d)

        self.alfa = A / (3 * B)
        self.s = B

        # Calculando a4 e a6 no campo K
        a4 = self.s**(-2) - 3 * self.alfa**2
        a6 = -self.alfa**3 - a4 * self.alfa

        self.EC = EllipticCurve(self.K, [a4, a6])

        # Pontos base convertidos para K
        self.Px = self.K(151122213495354007725011514095885315114540126930418572060461132839
        self.Py = self.K(463168356949264781694283940034751631413079938662562256157830336031

        self.L = ZZ(2**252 + 27742317777372353535851937790883648493)
        self.P = self.ed2ec(self.Px, self.Py)

        self.private_key = self.generate_private_key()
        self.public_key = self.generate_public_key()

    def generate_private_key(self):
        return randint(1, self.L - 1)

    def generate_public_key(self):
        return self.private_key * self.P

    def ed2ec(self, x, y):
        if (x, y) == (0, 1):
            return self.EC(0)
        z = (1 + y) / (1 - y)
        w = z / x
        return self.EC(z / self.s + self.alfa, w / self.s)

def sign(e, message):
    h = hashlib.sha512(message).digest()
    h_int = int.from_bytes(h, 'big') % e.L

    while True:
        k = randint(1, e.L - 1)
        R = k * e.P
```

```
        r = int(R[0]) % e.L
        if r == 0:
            continue

        k_inv = pow(k, -1, e.L)
        s = (k_inv * (h_int + r * e.private_key)) % e.L

        if s == 0:
            continue

        return (r, s)

def verify(e, message, signature):
    r, s = signature

    if not (1 <= r < e.L and 1 <= s < e.L):
        return False

    h = hashlib.sha512(message).digest()
    h_int = int.from_bytes(h, 'big') % e.L

    k_inv = pow(s, -1, e.L)
    u1 = (h_int * k_inv) % e.L
    u2 = (r * k_inv) % e.L

    R_prime = u1 * e.P + u2 * e.public_key

    return int(R_prime[0]) % e.L == r
```

E definindo a cifra e decifra baseado no que foi abordado anteriormente:

In [2]:

```
import numpy as np

def gen_keys(E, q):
    '''
    Gerar chaves pública e privada de ElGamal para a cifra e decifra a partir da ordem q
    '''
    sk = randint(1, q - 1)
    pk = sk * E.P
    return sk, pk

def encrypt(E,message,public_key):
    '''
    Escolher um número inteiro aleatório k no intervalo [1, L-1].
    Calcular o ponto C1 = k * P.
    Converter a mensagem m num ponto na curva elíptica
    Calcular o ponto C2 = m + k * pk.
    O texto cifrado é o par (C1, C2)..
    '''
    k = randint(1, E.L - 1)
    C1 = k * E.P
    m_point = encode_message(E, message)
    C2 = m_point + k * public_key
    return (C1, C2)

def decrypt(E,ciphertext,private_key):
    '''
    Usar a chave privada sk para calcular m = C2 - sk * C1.
    Converter o ponto m de volta para a mensagem original
    '''
    C1, C2 = ciphertext
    m_point = C2 - (private_key * C1)

    return decode_message(E, m_point)
```

```
def is_quadratic_residue(x, p):
    """
    Verifica se x é um resíduo quadrático módulo p.
    """
    return legendre_symbol(x, p) == 1

def encode_message(E, message, l=8):
    """
    Converte uma mensagem num ponto na curva elíptica.
    """
    K = E.K
    q = K.order()
    k = q.nbits()
    m_int = int.from_bytes(message, 'big')
    m_bits = m_int.bit_length()

    # Ajusta o tamanho da mensagem para k-1-l bits
    if m_bits > k - 1 - l:
        raise ValueError("Mensagem muito grande para o campo finito.")

    # Concatena a mensagem com l bits zero
    x = (m_int << l)  # Equivalente a m || 0^l

    # Tenta encontrar um ponto válido na curva
    for i in range(2**l):
        x_prime = x + i
        x_prime = K(x_prime)

        # Calcula x' = x^3 + a*x + b (equação da curva)
        y_squared = x_prime**3 + E.EC.a4() * x_prime + E.EC.a6()

        # Verifica se y_squared é um resíduo quadrático
        if is_quadratic_residue(y_squared, q):
            y = y_squared.sqrt()
            return E.EC(x_prime, y)

    raise ValueError("Não foi possível codificar a mensagem em um ponto da curva.")

def decode_message(E, point):
    """
    Converte um ponto na curva elíptica de volta para a mensagem original.
    """
    x = int(point[0])
    l = 8
    m_int = x >> l
    m_bytes = m_int.to_bytes((m_int.bit_length() + 7) // 8, 'big')
    return m_bytes
```

Exemplo de uso:

In [3]:

```
# Exemplo de uso
if __name__ == "__main__":
    # Parâmetros da curva Ed25519
    p = 2**255 - 19
    K = GF(p)
    a = K(-1)
    d = K(-121665) / K(121666)

    E = EcDSA_Ed25519(p, a, d)

    message = b"Hello, Ed25519!"

    try:
        m_point = encode_message(E, message)
        print(f"Mensagem codificada como ponto: {m_point}")
```

```
    except ValueError as e:
        print(e)

    decoded_message = decode_message(E, m_point)
    print(f"Mensagem decodificada: {decoded_message.decode()}")

    sk, pk = gen_keys(E, E.L)
    ciphertext = encrypt(E, message, pk)
    print(f"Texto cifrado: {ciphertext}")

    decrypted_message = decrypt(E, ciphertext, sk)
    print(f"Mensagem decifrada: {decrypted_message.decode()}")

    signature = sign(E,message)
    is_valid = verify(E,message, signature)
    print(f"Assinatura válida? {is_valid}")
```

```
Mensagem codificada como ponto: (962310367705108875825149657572689267 21 : 932739878228731826
505616693642967172524832614665368018602387705908566790822 3 : 1)
Mensagem decodificada: Hello, Ed25519!
Texto cifrado: ((127962028782744557204227064755988258824055164464257629466286163736866199439
41 : 105390118352535590982106358426971465241702002784252327152257345643151221758 40 : 1), (10
185570958684379981103720126097203433055520378810993774320897703052488232145 : 41111049837379
051909010216971895143489305887735831403316931163705349961246383 : 1))
Mensagem decifrada: Hello, Ed25519!
Assinatura válida? True
```

Transformar um PKE-IND-CPA em um PKE-IND-CCA

A transformação FO original constrói, a partir de (E_p, D_s), um novo esquema de cifra assimétrica (E'_p, D'_s), usando um "hash" pseudo-aleatório h de tamanho λ e um "hash" pseudo-aleatório g de tamanho $|x|$.

O algoritmo de cifra parametrizado pelos dois "hashs" h, g é

$$E'_p(x) \equiv \vartheta\, r \leftarrow \{0,1\}^\lambda \,.\, \vartheta\, y \leftarrow x \oplus g(r) \,.\, \vartheta\, r' \leftarrow h(r,y) \,.\, \vartheta\, c \leftarrow f_p(r,r') \,.\, (y\,,\, c)$$

O algoritmo D'_s rejeita o criptograma se detecta algum sinal de fraude.

$$D'_s(y,c) \equiv \vartheta\, r \leftarrow D_s(c) \,.\, \vartheta\, r' \leftarrow h(r,y) \,.\, \text{if } \; c \neq f_p(r,r') \; \text{ then } \; \perp \text{ else } \; y \oplus g(r)$$

Implementação

In [4]:

```
def g(r,message):
    """Hash pseudoaleatório g(r) com tamanho igual ao da mensagem x"""
    g = hashlib.sha512()
    if isinstance(r, int):
        r_bytes = r.to_bytes((r.bit_length() + 7) // 8, 'big')
    else:
        r_bytes = r
    g.update(r_bytes)
    final_hash = g.digest()
    while len(final_hash) < len(message):
        g = hashlib.sha512()
        g.update(r)
        final_hash += g.digest()
    return final_hash[:len(message)]

def h(r, y):
    """Hash pseudoaleatório h(r, y) com tamanho lambda_bits"""
```

```
    h = hashlib.sha512()
    if isinstance(r, int):
        r_bytes = r.to_bytes((r.bit_length() + 7) // 8, 'big')
    else:
        r_bytes = r
    ry = bytes(a ^^ b for a, b in zip(r_bytes, y))
    h.update(ry)
    return int.from_bytes(h.digest()[:8], 'big')
```

In [5]:
```
def f_p(E,public_key,r,rlinha):
    C1 = rlinha * E.P
    if isinstance(r,int):
        r_point = encode_message(E, r.to_bytes((r.bit_length() + 7) // 8, 'big'))
    else:
        r_point = encode_message(E, r)
    C2 = r_point + rlinha * public_key
    return (C1, C2)
```

In [6]:
```
def encrypt_FO(E, message, public_key):
    max_bits = E.K.order().nbits() - 1 - 8  # k = 255, l = 8 → 246 bits
    r = randint(1, 2**max_bits - 1)
    print("r (cifra):", r)

    y = bytes(a ^^ b for a, b in zip(message, g(r, message)))
    rlinha = h(r, y)
    c = f_p(E, public_key, r, rlinha)
    return (y, c)

def decrypt_FO(E, ciphertext, public_key, private_key):
    y, c = ciphertext
    C1, C2 = c

    r = decrypt(E,c, private_key)
    print("r (decifra):", r)

    rlinha = h(r, y)

    if c != f_p(E,public_key,r,rlinha):
        raise ValueError("ABSURDO")

    res = bytes(a ^^ b for a, b in zip(y, g(r, y)))
    return res
```

In [7]:
```
message = b"Ola!!!!!!!!"
try:
    m_point = encode_message(E, message)
    print(f"Mensagem codificada como ponto: {m_point}")
except ValueError as e:
    print(e)

decoded_message = decode_message(E, m_point)
print(f"Mensagem decodificada: {decoded_message.decode()}")

sk, pk = gen_keys(E, E.L)
ciphertext = encrypt_FO(E, message, pk)
print(f"Texto cifrado: {ciphertext}")

decrypted_message = decrypt_FO(E, ciphertext, pk, sk)
print(f"Mensagem decifrada: {decrypted_message.decode()}")

signature = sign(E,message)
is_valid = verify(E,message, signature)
print(f"Assinatura válida? {is_valid}")
```

```
Mensagem codificada como ponto: (245803384450832588717834324491 : 678019002550692275079761983
501007814753440762873943617362550938465920615188 8 : 1)
Mensagem decodificada: Ola!!!!!!!!
r (cifra): 111038085910335465676998667114447207859806939094428397776477970140073371180
Texto cifrado: (b'\xab_\xbc\xd4m\x9av\xd7\xa8\x12\xa8', ((165843930277138422321390568977249 1
284105184127814642394196245873355026834643 7 : 177432733319639807438913817433169766631231450 4
177544079423581424897144400412 1 : 1), (514032939336171799842471355321211788930656381105591 2 4
850285485959543254235745 6 : 232990850568161258300572927868046693739782062414998661857360237089
30732067324 : 1)))
r (decifra): b'>\xd8g\x83\x1e\xff\x8e\xe5\xf1\xba\x12\xa4\xe2b\xad\x14\x16\xa0S\xe8\xd5\xdd
\xcf\x1d\xe0kH\x1d\x91&,'
Mensagem decifrada: Ola!!!!!!!!
Assinatura válida? True
```

Exercício 3.b

"Oblivious Transfer" κ-out-of-n

O protocolo de "oblivious transfer" implementa um mecanismo de transferência de informação entre dois agentes: o **Provider** (também designado por Sender) e o **Receiver** (também designado por Adversário) . Em linhas gerais, o protocolo caracteriza-se da forma seguinte:

1. O **Provider** põe à disposição para comunicação futura n items de informação (ou mensagens) que ele enumera como $m_1, m_2, \cdots, m_n$ e que armazena de forma privada. Nesta fase a única informação tornada pública é o número de mensagens n .
2. O Receiver informa o **Provider** que pretende receber κ das n mensagens
3. Caso o **Provider** aceite o par (n, κ) os dois agentes, a começar pelo **Provider**, trocam uma sequência de mensagens e, no final,
   A. O **Receiver** passa a conhecer exatamente κ mensagens mas continua a ignorar o conteúdo de todas as restantes $n - \kappa$ mensagens.
   B. O **Provider** ignora a identificação ("is oblivious of") das κ mensagens que o Receiver passou a conhecer.

O protocolo usa um esquema PKE $\{(E_p, D_s)\}_{(s,p)\in\mathcal{G}}$ que neste caso irão ser a cifra e decifra do exerício 3.a..

Criterion

In [8]:
```
import numpy as np
from sage.schemes.elliptic_curves.ell_point import EllipticCurvePoint_field
class CknCriterion:
    def __init__(self, kappa, n, E):
        self.kappa = kappa
        self.n = n
        self.q = E.L
        self.seed = np.random.randint(0, 2**32)
        self.A = self.generate_A()
        self.u = self.generate_u()
        self.Fp = GF(self.q).unit_group()

    def generate_A(self):
        """Gera a matriz A usando XOF a partir da seed"""
        np.random.seed(self.seed)
        A = random_matrix(GF(self.q), self.n, self.n - self.kappa)
        return A

    def generate_u(self):
        """Gera o vetor u, que deve ser não nulo"""
```

```
        np.random.seed(self.seed + 1)
        u = vector(GF(self.q), [randint(1, self.q - 1) for _ in range(self.n - self.kappa)]
        return u

    def verify(self, p):
        """Verifica se p satisfaz o critério Ckn, ou seja, se p * A = u"""
        if len(p) != self.n:
            raise ValueError(f"p deve ter {self.n} elementos")

        p_values = []
        for x in p:
            if isinstance(x, EllipticCurvePoint_field):
                p_values.append(int(x[0]))
            else:
                p_values.append(int(x))

        Zq = GF(self.q)
        p_vector = vector(Zq, p_values)

        A_matrix = matrix(Zq, self.A)
        pA = p_vector * A_matrix

        u_vector = vector(Zq, self.u)
        return pA == u_vector

    def print_criterion(self):
        print(f"Matriz A:\n{self.A}")
        print(f"Vetor u:\n{self.u}")
```

Provider

In [9]:
```
class Provider:
    def __init__(self, pk, sk, n_mensagens,E):
        self.pk,self.sk = gen_keys(E,E.L)
        self.numero_de_mensagens = n_mensagens
        # Informação privada das mensagens:
        self.messages = [f"mensagem{i}" for i in range(n_mensagens)]
        self.criterion = None
    def define_criterion(self,kappa):
        n = self.numero_de_mensagens
        q = pk[1]
        self.criterion = CknCriterion(kappa, n, E)
```

In [10]:
```
n_mensagens = 100
# Parâmetros da curva Ed25519
p = 2**255 - 19
K = GF(p)
a = K(-1)
d = K(-121665) / K(121666)

E = EcDSA_Ed25519(p, a, d)

provider = Provider(None,None,n_mensagens,E)
print("self.pk,self.sk",provider.pk,provider.sk)
print("self.numero_de_mensagens",provider.numero_de_mensagens)
print("self.messages",provider.messages)
print("self.criterion = None")
```

```
self.pk,self.sk 685881669144742503854875481570988469539246500228449889805228953358758437031
(30602083431190442609414470087487728661183307130523708792658281652884793404333 : 567297216694
81270997740425657821911750968915534023101580171855114596804815157 : 1)
self.numero_de_mensagens 100
self.messages ['mensagem0', 'mensagem1', 'mensagem2', 'mensagem3', 'mensagem4', 'mensagem5',
'mensagem6', 'mensagem7', 'mensagem8', 'mensagem9', 'mensagem10', 'mensagem11', 'mensagem12'
, 'mensagem13', 'mensagem14', 'mensagem15', 'mensagem16', 'mensagem17', 'mensagem18', 'mensa
gem19', 'mensagem20', 'mensagem21', 'mensagem22', 'mensagem23', 'mensagem24', 'mensagem25',
'mensagem26', 'mensagem27', 'mensagem28', 'mensagem29', 'mensagem30', 'mensagem31', 'mensage
m32', 'mensagem33', 'mensagem34', 'mensagem35', 'mensagem36', 'mensagem37', 'mensagem38', 'm
ensagem39', 'mensagem40', 'mensagem41', 'mensagem42', 'mensagem43', 'mensagem44', 'mensagem4
5', 'mensagem46', 'mensagem47', 'mensagem48', 'mensagem49', 'mensagem50', 'mensagem51', 'men
sagem52', 'mensagem53', 'mensagem54', 'mensagem55', 'mensagem56', 'mensagem57', 'mensagem58'
, 'mensagem59', 'mensagem60', 'mensagem61', 'mensagem62', 'mensagem63', 'mensagem64', 'mensa
gem65', 'mensagem66', 'mensagem67', 'mensagem68', 'mensagem69', 'mensagem70', 'mensagem71',
'mensagem72', 'mensagem73', 'mensagem74', 'mensagem75', 'mensagem76', 'mensagem77', 'mensage
m78', 'mensagem79', 'mensagem80', 'mensagem81', 'mensagem82', 'mensagem83', 'mensagem84', 'm
ensagem85', 'mensagem86', 'mensagem87', 'mensagem88', 'mensagem89', 'mensagem90', 'mensagem9
1', 'mensagem92', 'mensagem93', 'mensagem94', 'mensagem95', 'mensagem96', 'mensagem97', 'men
sagem98', 'mensagem99']
self.criterion = None
```

Receiver

In [11]:

```python
import hashlib
from random import sample

class Receiver:
    def __init__(self, k, n_mensagens, q, E):
        self.k = k  # Número de chaves privadas geradas
        self.n_mensagens = n_mensagens
        self.q = E.L
        self.I = sample(range(n_mensagens), k)  # Seleção aleatória de k índices
        self.e = self.enumeration(self.I)
        self.p,self.s_values = self.generate_keys()
        self.s = self.generate_secret()
        self.tau = self.generate_authentication_tag()

    def enumeration(self, I):
        """Cria a função de enumeração que mapeia {1, 2, ..., κ} para os elementos de I ord
        I_sorted = sorted(I)
        return {i + 1: I_sorted[i] for i in range(len(I_sorted))}

    def generate_secret(self):
        """Gera um segredo aleatório (simulado como um número grande)"""
        return ZZ.random_element(2**32)

    def generate_keys(self):
        """Gera κ chaves privadas e publicas usando o genkeys de ElGamal com curvas elíptic
        vetor_pk = [0] * self.n_mensagens
        vetor_sk = []
        for i in range(1,self.k+1):
            sk,pk = gen_keys(E,self.q)
            vetor_sk.append(sk)
            vetor_pk[self.e[i]] = pk
        return vetor_pk,vetor_sk

    def generate_authentication_tag(self):
        """Gera a tag de autenticação hash(I, s)"""
        data = str(self.I) + str(self.s)
        return hashlib.sha256(data.encode()).digest()

    def complete_p_vector(self, A, u):
        """Completa o vetor p para satisfazer p * A = u no corpo finito Z_q."""
        Zq = GF(self.q)  # Define o corpo finito Z_q
```

```
        # Identificar os índices já preenchidos (valores diferentes de 0)
        filled_indices = [i for i in range(self.n_mensagens) if self.p[i] != 0]
        filled_values = vector(Zq, [int(self.p[i][0]) if isinstance(self.p[i], EllipticCurv

        # Criar a matriz A_filled (linhas correspondentes aos índices preenchidos)
        A_filled = matrix(Zq, [A[i] for i in filled_indices])

        # Criar a matriz A_empty (linhas correspondentes aos índices vazios, ou seja, onde
        A_empty = matrix(Zq, [A[i] for i in range(A.nrows()) if i not in filled_indices])

        # Calcular u' = u - (filled_values * A_filled)
        u_prime = vector(Zq, u) - filled_values * A_filled

        # Resolver o sistema linear A_empty^T * p_empty = u' no corpo finito Z_q
        try:
            A_empty_T = A_empty.transpose()
            p_empty = A_empty_T.solve_right(u_prime)
        except:
            # Se o sistema for singular, tentar solução alternativa (ex: mínimos quadrados)
            p_empty = A_empty_T.pseudoinverse() * u_prime

        # Preencher os elementos desconhecidos no vetor p (apenas onde p[i] == 0)
        empty_indices = [i for i in range(self.n_mensagens) if self.p[i] == 0]
        for i, idx in enumerate(empty_indices):
            # Gerar uma chave pública "má" usando gen_keys
            sk_mau, pk_mau = gen_keys(E, self.q)
            self.p[idx] = pk_mau

        return self.p

    def print_info(self):
        print("-------------------------------------------------------------------------")
        print(f"Seleção I: {self.I}")
        print("-------------------------------------------------------------------------")
        print(f"Função de enumeração e: {self.e}")
        print("-------------------------------------------------------------------------")
        print(f"Segredo s: {self.s}")
        print("-------------------------------------------------------------------------")
        print(f"Chaves privadas s_i: {self.s_values}")
        print("-------------------------------------------------------------------------")
        print(f"Vetor p (com chaves públicas mapeadas): {self.p}")
        print("-------------------------------------------------------------------------")
        print(f"Tag de autenticação τ: {self.tau}")
        print("-------------------------------------------------------------------------")
```

In [12]:
```
k = 20
receiver = Receiver(k,n_mensagens,provider.pk,E) # Receiver escolhe o conjunto I já na sua
```

Definidas as classes seguimos os passos igualmente como no exercício 1:

1. O **Provider** gera o critério C_{k,n} e envia-o ao **Receiver**

In [13]:
```
provider.define_criterion(k)
```

2. O Receiver escolhe um conjunto $I \subset \{1, n\}$, de tamanho $\#I = \kappa$, que identifica os índices das mensagens que pretende recolher.

Seja e a enumeração de I: a função crescente $e: \{1, \kappa\} \to \{1, n\}$ cuja imagem é I.

O Receiver compromete-se com a escolha de mensagens da seguinte forma (dado o conjunto I e a função crescente e):

1. Gera aleatoriamente um segredo $\mathbf{s}$ e , usando um XOF com $\mathbf{s}$ como "seed", constrói κ chaves privadas $s_1, \cdots, s_\kappa$.
2. Para cada $i \in \{1, \kappa\}$ gera chaves públicas $v_i \leftarrow \mathsf{pk}(s_i)$ e atribui o valor v_i à componente de ordem $e(i)$ do vector $\mathbf{p}$; ou seja , executa $\mathbf{p}_{e(i)} \leftarrow v_i$
3. Gera uma "tag" de autenticação para a seleção I e o segredo $\mathbf{s}$

$$\tau \leftarrow \mathsf{hash}(I\,,\,\mathbf{s})$$

(Definido na classe:)

In [14]:
```
receiver.print_info()
```

```
--------------------------------------------------------------------------
Seleção I: [50, 47, 83, 80, 60, 40, 91, 7, 10, 42, 75, 33, 45, 59, 92, 37, 65, 67, 28, 88]
--------------------------------------------------------------------------
Função de enumeração e: {1: 7, 2: 10, 3: 28, 4: 33, 5: 37, 6: 40, 7: 42, 8: 45, 9: 47, 10: 5
0, 11: 59, 12: 60, 13: 65, 14: 67, 15: 75, 16: 80, 17: 83, 18: 88, 19: 91, 20: 92}
--------------------------------------------------------------------------
Segredo s: 2503999058
--------------------------------------------------------------------------
Chaves privadas s_i: [594851661242538676068896213837216637541573193792073983048862919546877
516288, 593175075185422136920047211012753519303352848491041254010249394704942982310 2, 683712
042553110798360678708054619370667947744134574198003202702579655985975 0, 353961260752578899 67
265652583493210035374095933313033399694935456201780398 31, 274513418782103807957570832846023 5
617273866533207458379327576948614195441687, 400039477976107520330408630506804523743647491314
086048613256530336842116582 3, 369801715022214555775538294516681368980630220934356995704623 05
87256355886509, 272884752938458588960076496902570651640301642829363108324210398440694101639 2
, 490262914451723936116101035369167155438085292766082581172181342525859544 5593, 677251507823
351463914896937298832728268614465904219651021245067529729322919 5, 420100148594957338182859 75
723749098744276960798946505556763736777807701584 96, 245754853680203742563259787701202891453 0
697103372516230886982874813534715127, 331563995889166225180744508819973039765280777193610788
011025468153559727679 3, 102765819267066341075336548683369597267351872134591479721061953794 76
06565680, 716686550450573918346800966016220293649531851042483572298074232577139125307, 10484
909730913179787964780150596043467496822150212464998727927295832100010 89, 635021685568787800 1
040131685010078857539274565696697982962597774583431582730, 498049592035828193276268127023049
344864033037137698679752966194330030777930 6, 652021155244153783068404029859947456566850442 17
172134568644227198980981251 54, 256925555781205047017116596990675062716886914146371844317872 5
704897973537688]
--------------------------------------------------------------------------
Vetor p (com chaves públicas mapeadas): [0, 0, 0, 0, 0, 0, 0, (379418075829770628151704806 95
850709414656908904139673194061162179589851867622 : 513764124307213120891028547208363155114 83
286743892982202114681177606793478677 : 1), 0, 0, (495337066150445979120579314937482572758100
450267570821901484114228911737075 56 : 235175995259676633177880360739441492451429796611974148
409672718470554308401 90 : 1), 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, (427839147 0
694423362751745371781223689822032160635401735087042025262220992918 1 : 977938398268407982161 4
110609545278830016267913189202553557017025904900083260 : 1), 0, 0, 0, 0, (381791120504249574
463453206640030109074183557668244844275419377226896795637 40 : 149665052502578045737132176751
031988897376881783852662669419658759074309884 6 : 1), 0, 0, 0, (257873613061145798330311271 15
087866894240818199976395069713299145188646372565 : 113452841726672178234460627010798993228 59
947278017212124513206801433902471409 : 1), 0, 0, (554636759992870457056419122186816468833105
369413080240773095400201010318073 39 : 360288953629360340843884513159865358121312179509892286
707363314642104126001 40 : 1), 0, (176926917192521562219380565627710540577128926394098481542 7
0461621162710653297 : 477901007354369961318267192886521960369539012803910323145221287933681 4
9294732 : 1), 0, 0, (313441287873918776737782723309927746075329276036007014022253966239642 29
662396 : 424136931215643840837834237975855804866144117502307613075380879683160962468 89 : 1),
0, (207974427560841477981015259686857752349655280061549552804284297603316646155 21 : 50861143
777596692786781758882681379414475863686389289560504101127566605835475 : 1), 0, 0, (352269602
692386068320112549882692983826672230000329378273558109889419909736 05 : 316640977321853537024
621800276731419153055449720533710853351764568742977987 03 : 1), 0, 0, 0, 0, 0, 0, 0, 0, (4675
775989872930384990397774655554275600735287242896996924027879818585 1113439 : 2657410921446096
178739032103087737148405480094971128861569762959698658900765 9 : 1), (292239151006463839048 07
076652283559912691462194743125395876028463628313240 77 : 372755046202406864831800655564258898
513839322780658959041340629486059044508 33 : 1), 0, 0, 0, 0, (416651095429591066630081866566 7
266830010162153096319886526082988744376523146 2 : 418248846044922991822957728519478218588569 4
587909896715850806549184733332014 9 : 1), 0, (158088509625244430165443196723802573357055889 27
445715477960438561517718009674 : 368153913810096266724475114741272227001644033759930111173 88
04702824353972359 : 1), 0, 0, 0, 0, 0, 0, 0, (511505830789672423031378083416972411772561976 7
845813238251714440759311491040 0 : 293015555386122388673024231874691331493765280110605985290 0
0042273805005418978 : 1), 0, 0, 0, 0, (312292168347307444548296271143764345456003621563390 96
857337278121717246711233 : 269782549331547643299537674326119125865862339006717303767265685 24
15175974739 : 1), 0, 0, (130377109552636646923350334008870556484993596257296835703992363373 4
0386110231 : 389918166795470855511183653288078142109685974392717238549274977880381403872 52 :
1), 0, 0, 0, 0, (197304689849432223676723184589602240753141163274279451894592241819747454360
95 : 879950471740274931831642454236952833722796220841738336077848470763590622497 8 : 1), 0, 0
, (521497572600166859608944911797836775820574600845099621455796060421043190141 17 : 179900645
844598943574085971199997649482845440825881198427238509881789323897 79 : 1), (4079308141877894
113223334243732638831827033737240351531150226366279706172089 9 : 282679754819144883558924083 5
154090050807337537557283511384100749148936970490 9 : 1), 0, 0, 0, 0, 0, 0, 0]
```

```
----------------------------------------------------------------------
Tag de autenticação τ: b'\x01%\xca\xeb\xdd!\xf5gn>\xa7\xf7\x98TqDIJ\xdd\xb8\xf7\x90\xad\xbc
\xb3+b\xe4:\xb6\x16\xbd'
----------------------------------------------------------------------
```

Em seguida completa a definição de **p** atribuindo às compompentes $\{\mathbf{p}_j\}_{j\notin I}$valores tais que o vetor de chaves públicas **p** seja aceite pelo critério $\mathcal{C}_{\kappa,n}$.

Para que o vetor **p** satisfaça a equação $p \times A = u$, onde **A** é a matriz gerada pelo critério $C_{k,n}$ e **u** é o vetor correspondente, o Receiver precisa preencher os espaços em **p** que não foram preenchidos pelas chaves públicas de forma que a equação seja satisfeita. (Explicação mais a fundo no exercício 1)

In [15]:
```
completed_p = receiver.complete_p_vector(provider.criterion.A, provider.criterion.u)
tau = receiver.tau
```

3. O Provider determina $\mathcal{C}_{\kappa,n}(\mathbf{p})$; se **p** não for aceite pelo critério então aborta o protocolo.

Se **p** for aceite, então usa a variante IND-CCA da cifra IND-CPA com a "tag" τ

$$E'_p(x,\tau) \equiv \vartheta\, r \leftarrow \{0,1\}^\lambda \,.\, \vartheta\, y \leftarrow x \oplus g(r) \,.\, \vartheta\, r' \leftarrow h(r,y,\tau) \,.\, \vartheta\, c \leftarrow f_p(r,r') \,.\, (y\,,\,c)$$

cuja característica específica é o facto de se incluir o "tag" τ no "hash" $h(r,y,\tau)$ usado para construir a pseudo-aleatoriedade r'.

Usando esta cifra o Provider constrói n criptogramas

$$(y_i, c_i) \leftarrow E'_{p_i}(m_i)$$

com $i \in \{1,n\}$ que envia para o Receiver.

Iremos definir um novo "hash" pseudo-aleatório h para que consiga receber como argumento a tag τ:

In [16]:
```
def h_OT(r,y,t):
    """Hash pseudoaleatório h(r, y, t)"""
    h = hashlib.sha512()
    if isinstance(r,int):
        r_bytes = r.to_bytes((r.bit_length() + 7) // 8, 'big')
    else:
        r_bytes = r
    ry = bytes(a ^^ b for a, b in zip(r_bytes, y))
    ryt = bytes(a ^^ b for a, b in zip(ry, t))
    h.update(ryt)
    #full_hash = h.digest()[:16]
    return int.from_bytes(h.digest()[:8],'big')
```

E definir a cifra, que na realidade irá ser a do exercício 1.b. mas com um argumento extra τ:

In [17]:
```
def f_p_OT(E,public_key,r,rlinha):
    C1 = rlinha * E.P
    if isinstance(r,int):
        r_point = encode_message(E, r.to_bytes((r.bit_length() + 7) // 8, 'big'))
    else:
        r_point = encode_message(E, r)
    C2 = r_point + rlinha * public_key
    return (C1, C2)

def encrypt_FO_OT(E, message, public_key,tau):
    # Gerar r com tamanho máximo de 246 bits (k-1-l)
    max_bits = E.K.order().nbits() - 1 - 8  # k = 255, l = 8 → 246 bits
```

```
    r = randint(1, 2**max_bits - 1)
    print("r (cifra):", r)

    # Resto do código permanece igual
    y = bytes(a ^^ b for a, b in zip(message, g(r, message)))
    rlinha = h_OT(r, y,tau)
    c = f_p_OT(E, public_key, r, rlinha)
    return (y, c)
```

Determinamos então $C_{k,n}(p)$, se este for aceite ciframos todas as mensagens com as chaves públicas fornecidas em p:

In [18]:
```
import sys
from sympy import isprime
from sage.schemes.elliptic_curves.ell_point import EllipticCurvePoint_field  # Importação n

if not provider.criterion.verify(completed_p):
    print("-------------------------------------------------------------------------")
    print("O vetor p foi aceite")
    print("p final:[ ")
    for public_key in completed_p:
        print(public_key)
    print("]")
    print("-------------------------------------------------------------------------")
    ciphertext_vector = []
    i = 0
    for public_key in completed_p:
        if isinstance(public_key, EllipticCurvePoint_field):  # Verificação correta
            byte_length = (sys.getsizeof(provider.messages[i]))
            ciphertext_vector.append(encrypt_FO_OT(E, bytes(provider.messages[i], 'utf-8'),
            i += 1
        else:
            raise ValueError(f"encontrada chave pública inválida: {public_key}")
else:
    print("O vetor p não foi aceite. Abortado")
```

```
--------------------------------------------------------------------------
O vetor p foi aceite
p final:[
(3894686980605289708083715146422883652868544468724019183404702999228827714984 8 : 32691426237
2031355653710953677310279022902923526542397679404996714788518732528 : 1)
(4756296585006600962248087704620402987785259010744313563961430806906304894362 3 : 38190748760
4626783627830178501833677544046333332961704901780856251793066108648 : 1)
(2595773498899060902609659190654222316719134541779715218435988657679353402986 7 : 35167027615
0091041141337621533866795081112014191110281472052259466876497046548 : 1)
(5531809965129685235113636973290422985582905534550584201537034619057329502851 2 : 37110228132
6827300603188901033856179513517304472396576177661308514783067569 1 : 1)
(1383939464668927346287847865078712440401696689128764006191267085448919966208 0 : 54557458817
1808400920901393191019095684303916996322715869000277144624436193896 9 : 1)
(1555390437052717375538188957407672770023862672367880373290250507370681936967 1 : 41116493853
5720016501503452420879527311609613193919210368730039492073145356088 : 1)
(3468542324150482557334616774097323224785496658149693504364478097860201360678 1 : 24278623713
7176444176319838949634757134371641115434141388079467035569661158833 : 1)
(3794180758297706281517048069585070941465690890413967319406116217958985186762 2 : 51376412430
7213120891028547208363155114832867438929822021146811776067934786776 7 : 1)
(5007592232758871704870884534843237707169256262627827923619537433171422002604 2 : 46529626644
0139949423134159286410883749536653924145997020880616372125526293696 9 : 1)
(3813349096780881310727075516266269632677180992887011552779561798226217205662 9 : 16865359420
2429407943804033224861268503359375212083757570785874377860100784688 : 1)
(4953370661504459791205793149374825727581004502675708219014841142289117370755 6 : 23517599525
9676633177880360739441492451429796611974148409672718470554308401900 : 1)
(1828493612335342381383459064467413347409409076262470191435062261155613601286 8 : 34636058025
4624662013221256352073139272657514505186664419492263818878878489310 : 1)
(1024230085373206874920819844783287150637485407503672682944660162972152516600 6 : 56541526497
9676611328434388542058514045914002395219788366492857576117490659488 : 1)
(2052698457823174762256011026296371699682217014527994307487787812646974267574 7 : 82572654088
2933323419294519148847208972357073739990540236306687514756627525858 : 1)
(5179998185382299783665700956426775238818082082442651776441811707397483804873 7 : 22432603622
8795344928110076400707754708947973812378781612679762217679756638260 : 1)
(2706306922025758218382143756587100091824251049983661516067758230608097718148 5 : 65988031478
4452861449886569171525026493653817794661562370420831852663330116788 : 1)
(4802321027290326082491774524789752804669713589612680510500980269870490664381 6 : 49245651293
2548711824298052135103463133848871638733962409522811868459911321108 : 1)
(5694977990464457166693791216078633962599395717691810310625255291156082067152 : 390883516842
5230845528767858285885211275017059978734581942253833063163820046688 : 1)
(2957581558655380595232203684073895235351106441601102566506545231510885140290 8 : 20997939700
4874837451736392228139610954019758464822456954757998666669568799618 : 1)
(5617284818167614218833629642591648744652921790565075753680381574039989205723 8 : 57811166412
2506950904599522308467665921819195258883331505992786953472080162318 : 1)
(2056005218330856634751163510539923967958475490536781478386359025306537774025 9 : 29644937046
2610962966944817237836938851064160677380547592764700798249907226278 : 1)
(3018612731200514939715968724191976503188554151466830880300227880111030204652 5 : 37271350692
6853044225623272366349919789825469798468624260559229078664822665408 : 1)
(1099129417321426525528753856781614027859746750286512469084092327214482865129 1 : 52162014802
4626577788909109792693820996785513531256731524771831464490100891168 : 1)
(1614060097763633273760714452329557739581731344055890165451852950582691804856 5 : 41677613620
9603566010371242376866747652185209040356020521780729513136584095448 : 1)
(4349364526151143925724477393663795625499741630137366532194000233890088187423 9 : 48057956181
5370567189374275045963043816554907620099749966504871798787794223548 : 1)
(2496714145310385569549517220993480933270601884497759351248374440585665575088 5 : 25702491795
7121673971128650692718896034913309731317277502656199339097861067088 : 1)
(2605359077315767466835274350143047483943532229626731503818221506291379342319 9 : 12068254196
4193739544109651105936705748555311463136790138321270895721647400498 : 1)
(3410505238150724006840708321406119444682058661089257215642745752250921539812 2 : 25413560123
7514810711135820239751809771690854926941285047731468453839170339488 : 1)
(4278391470694423362751745371781223689822032160635401735087042025262220992918 1 : 97793839826
8407982161411060954527883001626791318920255355701702590490008326088 : 1)
(4919306853067560592984488025682741165069442283755016869418437415980443009075 1 : 64764279994
7511361604426428878267180587844311769604673608579662187210625501388 : 1)
(9046163262722287090678498139154303037417076339406064197838024634729167030008 : 307720541998
6603725899330194400418006951066532894048035515657216782232880739188 : 1)
(4690748894286234978369205584971669942865885093884571591885976007374465312181 6 : 33497330679
```

51156518922366102937527341153158074877099215706993647857217332980 2 : 1)
(21770960897317068221725301626629721364058238826712145758699293949329222027008 : 24042889019
04044521660668598206421091949344570316574678283068664344635988128 : 1)
(38179112050424957446345320664003010907418355766824484427541937722689679563740 : 14966505250
25780457371321767510319888973768817838526626694196587590743098846 : 1)
(22990093330676584098645155602777173827071329106838641318614753863779737525297 : 25842952676
66463877131076526159509618609968674534650574800299666088520195503O : 1)
(17229558500089344672231949660290467822199136710594797740439721736687532975752 : 45366324241
44070347008713192321812263613588089083208047204436880599714212684 8 : 1)
(15653797858731747247532885806270634838129211017619693817522448545660313518415 : 23381750450
10123087002025397601330296258996981291521550811122188499161070966 8 : 1)
(25787361306114579833031127115087866894240818199976395069713299145188646372565 : 11345284172
66721782344606270107989932285994727801721212451320680143390247140 9 : 1)
(55692958199901330565479969958054356384668492479461114501983210270026247227335 : 17245951951
14084253978385254402907514300327036417702975650778160337291723185 4 : 1)
(10092819763926598687362423188703459719726787997681649153783111130980234384837 : 46783136121
24029307239969290024142080203750424829433040746801304493050024449 5 : 1)
(55463675999287045705641912218681646883310536941308024077309540020101031807339 : 36028895362
93603408438845131598653581213121795098922867073633146421041260014O : 1)
(28097868380558991924572294325059078909258474070803594164136998055408590502975 : 12778145046
62629146218249750451614309325414561965687727783536553637294012703 3 : 1)
(17692691719252156221938056562771054057712892639409848154270461621162710653297 : 47790100735
43699613182671928865219603695390128039103231452212879336814929473 2 : 1)
(48584902444560289827297652834384990913831736776772620508599470665411047512777 : 63857719998
88394045525768953369354926228385156660524369573000598744741198004 : 1)
(43198069199128391589808428771095329778119505767172327428950838672634516718 97 : 463739730351
52505845167003502732014868109647713845809749783326334441827018975 : 1)
(31344128787391877673778272330992774607532927603600701402225396623964229662396 : 42413693121
56438408378342379758558048661441175023076130753808796831609624688 9 : 1)
(55686012684654786237244153772103962460194962228836731660178624918784461924 95 : 439600097300
92717427440930936912789118441191984176611121699607803148296261901 : 1)
(20797442756084147798101525968685775234965528006154955280428429760331664615521 : 50861143777
59669278678175888268137941447586368638928956050410112756660583547 5 : 1)
(47781441884773949457651838591312366008821607232210800040953849871710308002810 : 30060045467
33920347056922814322243564501959141553637684138828045230500609322 0 : 1)
(59105421611098074582978658778801438400683770007727583941937565997831231945 13 : 257009217396
34134064362266286914587055777395992034643020385720152182187188 62 : 1)
(35226960269238606832011254988269298382667223000032937827355810988941990973605 : 31664097732
18535370246218002767314191530554497205337108533517645687429779870 3 : 1)
(38149712408432665093803740994129972896188129201059458766462348339728857508355 : 27854040183
80702649378678737089480090595503213140044530902269159015789905032 2 : 1)
(31331119329594501800217117935595760402342062051345417071411412949511244482279 : 37784459863
34183771925451294312784647999386266786995730471943969794469981098 6 : 1)
(46277208397205236220297324491129529712899166656619702663707881846932849746556 : 55080714120
76560800017545806370535786523029755667264835518765040498216058478 7 : 1)
(31778689419525629215883390491527392690329243288318773906879560319152597291213 : 53744266619
16236189713029328772958628054284100798090443814777539075274069277 0 : 1)
(54799717427632583216631325887845775333510374791400588100086400472252817233314 : 38839595863
02732446720854681921672477981399569098840481747408988668624932926 : 1)
(24119355566860721786354524066375074067859814493255719433001085601534107829872 : 93667615522
67611788243946907270843029255152733066034916803419074839369192439 : 1)
(30039704661770068473760483105953355866794002322912677560855619940212603185 19 : 406776711226
56356862473022299737960536642236318932836977863177165958005189308 : 1)
(11554138206826118882577496451563411607578615528825438393396118020271241951009 : 39685899874
98013072013633046861219902139378874342942297731296345737211328537 5 : 1)
(46757759898729303849903977746555542756007352872428969969240278798185851113439 : 26574109214
46096178739032103087737148405480094971128861569762959698658900765 9 : 1)
(29223915100646383904807076652283559912691462194743125395876028463628313240 77 : 372755046202
40686483180065556425889851383932278065895904134062948605904450833 : 1)
(58140061579846439626781675508579951026111163872613206319485311362177194983 94 : 161022103808
35738653735730897808537396910825034161937953669821061205750861964 : 1)
(33785958560069733503719955810062777180641375366072954968868913297259302428 41 : 243869445456
40599129014255264754191415338883682911343551979503471232385913712 : 1)
(48016751827914877223566580989640666291822847086120774506125611275033696505265 : 47298892710
27299238381871106154044250563167594137267791788649911297492149949 9 : 1)
(19963514439028153332672235761329513385860314013382932918977721065799725421 71 : 390899014961

770704012997887145301524241230555372785884740149529175398778955 6 : 1)
(41665109542959106663008186656672668300101621530963198865260829887443765231462 : 41824884604
49229918229577285194782185885694587909896715850806549184733332014 9 : 1)
(30536258996373343361854260947420801351880703186720929897536912512654566825524 : 84379158942
57689152301525169277416252011112702854863104648589177339615729525 : 1)
(15808850962524443016544319672380257335705588927445715477960438561517718009674 : 36815391381
00962667244751147412722270016440337599301111738804702824353972359 : 1)
(17950426427629199280403960361482374860491282090383967885069162358268139072107 : 62658736035
23759421146069455852000069424311322892570397145962210518355726617 : 1)
(42669682039266354655260027096139731589033564812145730334073005820447765883 6 : 5203989413536
93148467923585603372157287865816153926028849279854350459302732 58 : 1)
(41648604345127218090284883944956316224300001672119060143656439042703138099281 : 41780611726
41691816809944311655342373536514592671122545217634600688739635560 0 : 1)
(27684129251420845369712556902799015530226042512829503888734845680111872028900 : 26483984359
25563574622798938915690624336375318098371298211321604433848526088 4 : 1)
(13342164333982979107445545896895460361278145165561119332340452400869669042201 : 13585176351
51694829121862450918420323745933000275018486026505104066094021655 3 : 1)
(49287517424545587553809356380384344519269394565123327639944195670698140476904 : 14187761806
40337342185563053028223348373905386869406192924896297139289247681 0 : 1)
(18645863915047582730281013868174803796399964859555466095011830841531969352928 : 59745853068
79996916632212596560885915179844290617345853125097997085539765137 : 1)
(51150583078967242303137808341697241177256197678458132382517144407593114910400 : 29301555538
61223886730242318746913314937652801106059852900004227380500541897 8 : 1)
(44502537382369284797081758347842993081364145503821172609201561437973667264133 : 52262068692
76036777734263206925442293774020896860240373161139377060171311556 1 : 1)
(29386810092007951991084600401210786612637307310194651701824507497310913722765 : 32495723111
28056559640882558402049635068677968336930956972694943716850015773 1 : 1)
(55567150705692742701627830683296172131705409143626637941309647707181498327608 : 77929512381
52559303123069294389851360569347306276532978350895747875704939561 : 1)
(19720656975886193044211492061474012549693963529114856340719434453360131506106 : 22448175128
35793109303637519640793235544209648020988601409427118142565109986 1 : 1)
(31229216834730744454829627114376434545600362156339096857337278121717246711233 : 26978254933
15476432995376743261191258658623390067173037672636852415175974739 : 1)
(38900847109852151359033117089506004271182481016224568436681361217528463514 30 : 227568579890
37508535298987786173444726576851961092141942406141463000563587910 : 1)
(48880350059101966472169746601649482791429405290134594150425515682561785594386 : 19932302527
79768077466881707964141326261929976940919192862350485593258657763 : 1)
(13037710955263664692335033400887055648499359625729683570399236337340386110231 : 38991816679
54708555111836532880781421096859743927172385492749778803814038725 2 : 1)
(21646538797775092735508682901198102145607950321213068218565196246264259924984 : 65979032934
91466222651281677516612090499760415760711757761446742364258768796 : 1)
(42144416660398431886806051804712651140923183072767568184399318673707790551626 : 12456535826
32746124330204518467623360879490285605731777225278989991377269191 2 : 1)
(29097233832517032939081795709544651691601408008297681548895216563145436694 65 : 379690487469
51145279493039320730131300447752022965247232990073443962388218378 : 1)
(36061094882866097390305231654180762507777922822943849464180283512300044569657 : 21913841323
76141800770596385038320556946726232910446230146463964402190447882 : 1)
(19730468984943222367672318458960224075314116327427945189459224181974745436095 : 87995047174
02749318316424542369528337227962208417383360778484707635906224978 : 1)
(52745098794351458913491956367246907885793990442682203502438879778588398663268 : 27890191633
50676603306477962007671604048601374996402514020677952984576008836 1 : 1)
(32116615703600201403950356728504648290136838129174136183190515817303165928603 : 19237200153
16077658701483052307176808773351437353656961713722280964137371165 6 : 1)
(52149757260016685960894491179783677582057460084509962145579606042104319014117 : 17990064584
45989435740859711999976494828454408258811984272385098817893238977 9 : 1)
(40793081418778941132233342437326388318270337372403515311502263662797061720899 : 28267975481
91448835589240835154090050807337537557283511384100749148936970490 9 : 1)
(28108980298002068275904928571542916693700445517677151666314117946019231735436 : 55013112869
16965281031712852800049405978448795243238730118658944879563286100 : 1)
(32548294000497332381506109908023420029326013443643596025854635933495166429352 : 47524041081
62891089660752379751150699115855779259708733072205295400597159187 : 1)
(29456756856794826368022389279615435818740604387260513036816680555578093171364 : 10400742080
58417565864315399536125761183231603550973866445097929615856155108 7 : 1)
(21650195750328367773664817793196188280985377130901225340760066971596405132832 : 92568082011
59630784409384962580741205849202476149019902091744117157879999390 : 1)
(49082993858602415664227363169220998029244033829700255725450474449319281321517 : 18303392675

```
3050117539184157148872276330624817215380505970732718630709272110 81 : 1)
(8085212442357625640382330708950364505416528523305527330519580650052218284151 : 318081244390
059417462873823465670883289167430904870753190960617353183791651 16 : 1)
(7301432745040459513464350394942581802920536787354667693662324422192036645837 : 359359649631
143646415337713040311551814315263693022306748293594391115960009 55 : 1)
]
---------------------------------------------------------------------
r (cifra): 22335710773898632071551093468263494858944130387203556992597626008742048717
r (cifra): 34564592087978190911594063524916631884705808678629902124120615062678149757
r (cifra): 10939077783957575204298147255094954877133395089469806072678886604540424869 5
r (cifra): 62613766643702320771376522251543840103064103773368576084477601144442596937
r (cifra): 43111361074034688688093315560314299447270367138376280597971232700733183192
r (cifra): 80333546613658826172745301190226444887361858162516242774209949774833185628
r (cifra): 11300538838174958456633216452357338657537597023129055588399696390380003786 3
r (cifra): 49149662556324535502800936075703990945573320462148245988804483574425468210
r (cifra): 92384333539376009068125061184141567910180342690854832449459614318266188485
r (cifra): 89385587816682956516966582916311066808513152890269043038911278540633362833
r (cifra): 81284620983733203859764411630976408835696504998241316016482128040392503376
r (cifra): 47330671819666860152949331408561148661276764889558301463172681208198935169
r (cifra): 39198517091759693930100423036964575079322133481787276288090912493824827 6
r (cifra): 92798612503714114735060816115911596867175172129146251007176632042971030041
r (cifra): 32801083720466736682613855691562868278655109728555633764135940479520671490
r (cifra): 88270779781760126001353091271030555224703289687350504518135449229417310446
r (cifra): 50130811856567403929204408173833186415563584346287536501911260541222225588
r (cifra): 69219309818947268013013027948726096029104288688296587813801503448695234589
r (cifra): 75634306707476243614555061727305757978178261268422421298687505830165131 14
r (cifra): 39960329840294079172140676546437959629431264402104030069540770656347960396
r (cifra): 11302822280049945486831324255667615340325802055040730096872285542517847373 8
r (cifra): 43732435558697675269548169782838772212905830250854969919328308301850518307
r (cifra): 79520086275892905088410615811324264087638231301728762775340340016952204598
r (cifra): 25848413819609707277144514032909423459280586213254107527251484321640800671
r (cifra): 10462542574728808135252971677597705837761605616263311605198997684406929061 9
r (cifra): 33709138434540088522185140193672643489763586260274307794977247510677360879
r (cifra): 23383383339135641539084097933052048995157830524918205690744221705424811 65
r (cifra): 46059591058385314266967230609122281216668979580463813602776375203994389412
r (cifra): 76076836847146477064976496981079887652947749045998614872609620988251997472
r (cifra): 10752383329922128986185562242316225640564837096116693547886897781116472864 1
r (cifra): 78037785121343326157043202959610883156416719761168827318140354511052118000
r (cifra): 63564465874452659415914364984152808554163994313930017860694089440460616 63
r (cifra): 13226319494620249139741691363521063343377548318973249935269291573962477652
r (cifra): 24730599349107935803071978744651709852531077565147579446409456907425627621
r (cifra): 20695507754560783824472749123452894379719006197398969693425116024141735030
r (cifra): 29435222493113793553824662285703600227632885886658929462935051698570645166
r (cifra): 53636148065298955782044215101606372145779602176777326971203173940819214352
r (cifra): 88068975173869506843226959529129124590486684049393011743359414813488574690
r (cifra): 38985183064034257511910299329830139286003323528456243358202409107098967892
r (cifra): 40807026953442673016647184622423245557763085519525933935194186939067674117
r (cifra): 10792679121311922084731662667617155949682806511472184703847931111697294303 6
r (cifra): 50489335014020892491401343112625477627231712638655379664894018943701667432
r (cifra): 11189175950482677243324890449637330807636711485802003399788314969902904223 5
r (cifra): 26487393668471421169631223457879473927917180042283619084384897697122789467
r (cifra): 10494492444719585815995313179673838725051215457829924377148811179681903830 0
r (cifra): 27403319417963750290300176043315373458185273872650146436920195615723824382
r (cifra): 16347826298774673237899894975047218074350318069431458711438801753501130266
r (cifra): 59872528972588361107604401804854937966989291557711236395193366477651841 19
r (cifra): 10553807962104639906596504361889965033241932432573970690514092451882795649 6
r (cifra): 84265314893969194277921120243277339641500230932338582313800757761431920205
r (cifra): 10685924871797154337855520891775592684348477678002881682809401922860346430 9
r (cifra): 95784457198291302783134464458787021786177443841326125813084609780170806513
r (cifra): 73703368572538397257133323772702346619665594954468947950342733561852268738
r (cifra): 10967112490663924608056558550994068461932623902882427454016204057869212399 2
r (cifra): 65792651019018033432135101388665364057097798695656043195020482313430896735
r (cifra): 42327296932151006353227215981320748952470333014352901250605853848365696004
r (cifra): 53219428720149135790722823375695807273296920142863916998554923941728151664
r (cifra): 20007949107861202543621293520558025356093573736930196680370051800516463 18
r (cifra): 49957914097566644186909872758217020540175267428409998335836069285239751373
```

```
r (cifra): 5846446557769933184581547317168522972198529987124900220051124609596702235 4
r (cifra): 2259874172838699289926927788289008097982530849956137335022308343041159198 2
r (cifra): 6598847190197150479497930325190546331317380932504802877790037992314056428 7
r (cifra): 8414512770270558844589969213075601912253188344219736007943463777976569582
r (cifra): 1208514404084361379189638012633875176058093866807373072969306511488648691 7
r (cifra): 5812585298205092618086145009008612623668783927640413282657681733144286124 9
r (cifra): 5484765706799189085081120534066320890011743094373253421679875352171078281 7
r (cifra): 5902660842054863240775909932370667277821126074353778156180671590330354403 6
r (cifra): 5011466109196774707266810144121451856200705906128669377674499819931365361 7
r (cifra): 4040832764987829891918184775577181012402338872649724909207622591745000984 9
r (cifra): 2232871273685342427142537201910795189474958950374677500345673166773374951 9
r (cifra): 3821150570822171155590093954299189858636031695051908241516283000370515916 0
r (cifra): 5704255649095268924704123574223265858641191646487194645522212348824073745 7
r (cifra): 5420610021651760202970624030180688842715951849480766487824652690754957804 2
r (cifra): 3871128534315813000739545548325192236599208006957624897211892816617931967 6
r (cifra): 7133001018987881649059609393205796035230355688108210147880437757474141268 5
r (cifra): 6239524851357555169839750974192714299200616405930618727880671989900029329 0
r (cifra): 4504537886730728037864637620467830606738356719469415884186257244755946555 9
r (cifra): 5798421158498400039409653080512144574617057344342537784084393307700373730 0
r (cifra): 9594779400492414147069115272522359746551643837513646236086173799919430398 2
r (cifra): 3598314190525171472699410984973150018035741092273067115127508547545129496
r (cifra): 8457396366260858095703767510392207644706296007729921919086112847305463552 5
r (cifra): 5226048653842371002193780412022194582322284420778601014197949185294579364 9
r (cifra): 5173654584504075579719709166761626557787262769996662260830682941742097566 0
r (cifra): 4771021520988632112753962190931016218164864427636644045235174882002570558 8
r (cifra): 6843443752793144396785597107873690652631087383356827163681108598604149460 2
r (cifra): 1607333000870443930872842074171962782276666768404250352582196426417428123 7
r (cifra): 7869234943080990789848242558301895340254090952460372896296446140099277746 5
r (cifra): 2675620712343035380793287938986876220642911445796224725373679909602203220 0
r (cifra): 7990253387251869039410904097210902448091364218808918847206453550577534779 7
r (cifra): 3506469925039860695527806558874587111476942451935871955090418708344111646 3
r (cifra): 9049895890058887731041042699922949397016471151036779238793914996945349743 5
r (cifra): 8800194358549456925693953906447442306546792624164751922623220577354119892 3
r (cifra): 1011074837664470985527799938773905640609258869920444111383122311722970216 39
r (cifra): 7578888924222791801137343058612550972877345893356396783096617862355687520 1
r (cifra): 2853521222307516928386985163004064417474239385646012645821705463684855227 2
r (cifra): 7968517804453387562401283938196038211354735638256942389909360501361881899 2
r (cifra): 9953671212955339608207114134062493671979136890267450265931695352936224348 9
r (cifra): 8781465172704284160917785463109129000708095830739503153594794821110968657 7
r (cifra): 1613967758131420268610855858137137089773607177415547162604775850189513940 5
r (cifra): 1084621025168195318568516379608209477992423444653949866099691064939133819 50
```

4. O Receiver usa a variante IND-CCA da cifra IND-CPA com a "tag" de autenticação τ

$$D'_s(y,c,\tau) \;\equiv\; \vartheta\, r \leftarrow D_s(c)\ .\ \vartheta\, r' \leftarrow h(r,y,\tau)\ .\ \textsf{if}\ \ c \neq f_p(r,r')\ \ \textsf{then}\ \ \perp\ \textsf{else}\ \ y \oplus g(r)$$

uma vez mais a única característica particular deste algoritmo é o uso da "tag" de autenticação τ na construção da pseudo-aleatoriedade $r' \leftarrow h(r,y,\tau)$.

O agente Receiver

- conhece, porque criou, a "tag" τ que autentica o conjunto de mensagens escolhidas I e o respetivo conjunto de chaves públicas (as "boas" chaves). - conhece , porque gerou e armazenou num passo anterior, as chaves privadas s_i para todos $i \in I$
  - conhece, porque recebeu do Receiver, todos os criptogramas $\{(y_i, c_i)\}_{i\in\{1,n\}}$

Então, para todo $i \in I$, pode recuperar a mensagem

$$m_i \;\leftarrow\; D_{s_i}(y_i, c_i, \tau)$$

Como na cifra vamos fazer com que a decifra possa receber a tag de autenticação:

In [19]:
```python
def decrypt_FO(E, ciphertext, public_key, private_key,tau):
    y, c = ciphertext
    C1, C2 = c

    r = decrypt(E,c, private_key)

    rlinha = h_OT(r, y, tau)

    if c != f_p(E,public_key,r,rlinha):
        raise ValueError("ABSURDO")

    res = bytes(a ^^ b for a, b in zip(y, g(r, y)))
    return res.decode('utf-8')
```

In [20]:
```python
# Decifrar e verificar as mensagens
decrypted_messages = []
print(len(ciphertext_vector))
for msg_number, idx in receiver.e.items():
    if idx < len(ciphertext_vector):  # Verifica se o índice é válido
        print(f"\nNúmero da mensagem: {msg_number}")
        print(f"Índice no vetor ciphertext_vector: {idx}")
        print(f"Chave privada (sk): {receiver.s_values[msg_number - 1]}")  # Ajuste para ín
        print(f"Chave pública (pk): {receiver.p[idx]}")
        # Verifica se a mensagem foi cifrada corretamente
        if ciphertext_vector[idx] is None:
            print("idx:",idx)
            print(f"Mensagem {idx} não foi cifrada corretamente.")
            continue

        y, c = ciphertext_vector[idx]
        sk = receiver.s_values[msg_number - 1]
        decrypted_message = decrypt_FO(E, (y,c), receiver.p[idx],sk, tau)

        if decrypted_message is not None:
            print(f"Mensagem decifrada: {decrypted_message}")

            # Compara com a mensagem original no Provider
            if decrypted_message == provider.messages[idx]:
                print("Decifração bem-sucedida! A mensagem decifrada corresponde à original
                decrypted_messages.append(decrypted_message)
            else:
                print("!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!")
                print("Erro na decifração: A mensagem decifrada NÃO corresponde à original.
                print("!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!")
        else:
            print("Erro na decifração intencional.")
```

100

Número da mensagem: 1
Índice no vetor ciphertext_vector: 7
Chave privada (sk): 5948516612425386760688962138372166375415731937920739830488629195468777516288
Chave pública (pk): (37941807582977062815170480695850709414656908904139673194061162179589851867622 : 5137641243072131208910285472083631551148328674389298220211468117760679347867 7 : 1)
Mensagem decifrada: mensagem7
Decifração bem-sucedida! A mensagem decifrada corresponde à original.

Número da mensagem: 2
Índice no vetor ciphertext_vector: 10
Chave privada (sk): 5931750751854221369200472110127535193033528484910412540102493947049429823102
Chave pública (pk): (4953370661504459791205793149374825727581004502675708219014841142289117 3707556 : 2351759952596766331778803607394414924514297966119741484096727184705543084019 0 : 1)
Mensagem decifrada: mensagem10
Decifração bem-sucedida! A mensagem decifrada corresponde à original.

Número da mensagem: 3
Índice no vetor ciphertext_vector: 28
Chave privada (sk): 6837120425531107983606787080546193706679477441345741980032027025796559859750
Chave pública (pk): (4278391470694423362751745371781223689822032160635401735087042025262220 9929181 : 9779383982684079821614110609545278830016267913189202553557017025904900083260 : 1)
Mensagem decifrada: mensagem28
Decifração bem-sucedida! A mensagem decifrada corresponde à original.

Número da mensagem: 4
Índice no vetor ciphertext_vector: 33
Chave privada (sk): 3539612607525788996726565258349321003537409593331303339969493545620178039831
Chave pública (pk): (3817911205042495744634532066400301090741835576682448442754193772268967 9563740 : 1496650525025780457371321767510319888973768817838526626694196587590743098846 : 1)
Mensagem decifrada: mensagem33
Decifração bem-sucedida! A mensagem decifrada corresponde à original.

Número da mensagem: 5
Índice no vetor ciphertext_vector: 37
Chave privada (sk): 2745134187821038079575708328460235617273866533207458379327576948614195441687
Chave pública (pk): (2578736130611457983303112711508786689424081819997639506971329914518864 6372565 : 1134528417266721782344606270107989932285994727801721212451320680143390247140 9 : 1)
Mensagem decifrada: mensagem37
Decifração bem-sucedida! A mensagem decifrada corresponde à original.

Número da mensagem: 6
Índice no vetor ciphertext_vector: 40
Chave privada (sk): 4000394779761075203304086305068045237436474913140860486132565303368421165823
Chave pública (pk): (5546367599928704570564191221868164688331053694130802407730954002010103 1807339 : 3602889536293603408438845131598653581213121795098922867073633146421041260014 0 : 1)
Mensagem decifrada: mensagem40
Decifração bem-sucedida! A mensagem decifrada corresponde à original.

Número da mensagem: 7
Índice no vetor ciphertext_vector: 42
Chave privada (sk): 3698017150222145557755382945166813689806302209343569957046230587256355886509
Chave pública (pk): (1769269171925215622193805656277105405771289263940984815427046162116271 0653297 : 4779010073543699613182671928865219603695390128039103231452212879336814929473 2 : 1)
Mensagem decifrada: mensagem42
Decifração bem-sucedida! A mensagem decifrada corresponde à original.

Número da mensagem: 8

```
Índice no vetor ciphertext_vector: 45
Chave privada (sk): 272884752938458588960076496902570651640301642829363108324210398440694101
6392
Chave pública (pk): (313441287873918776737782723309927746075329276036007014022253966239642229
662396 : 424136931215643840837834237975855804866144117502307613075380796831609624688 9 : 1)
Mensagem decifrada: mensagem45
Decifração bem-sucedida! A mensagem decifrada corresponde à original.

Número da mensagem: 9
Índice no vetor ciphertext_vector: 47
Chave privada (sk): 490262914451723936116101035369167155438085292766082581172181342525859544
5593
Chave pública (pk): (207974427560841477981015259686857752349655280061549552804284297603316641
615521 : 508611437775966927867817588826813794144758636863892895605041011275666058354 75 : 1)
Mensagem decifrada: mensagem47
Decifração bem-sucedida! A mensagem decifrada corresponde à original.

Número da mensagem: 10
Índice no vetor ciphertext_vector: 50
Chave privada (sk): 677251507823351463914896937298832728268614465904219651021245067529729322
9195
Chave pública (pk): (352269602692386068320112549882692983826672230000329378273558109889419990
973605 : 316640977321853537024621800276731419153055449720533710853351764568742977987 03 : 1)
Mensagem decifrada: mensagem50
Decifração bem-sucedida! A mensagem decifrada corresponde à original.

Número da mensagem: 11
Índice no vetor ciphertext_vector: 59
Chave privada (sk): 420100148594957338182859757237490987442769607989465055567637367778077015
8496
Chave pública (pk): (467577598987293038499039777465555427560073528724289699692402787981858 51
113439 : 265741092144609617873903210308773714840548009497112886156976295969865890076 59 : 1)
Mensagem decifrada: mensagem59
Decifração bem-sucedida! A mensagem decifrada corresponde à original.

Número da mensagem: 12
Índice no vetor ciphertext_vector: 60
Chave privada (sk): 245754853680203742563259787701202891453069710337251623088698287481353471
5127
Chave pública (pk): (292239151006463839048070766522835599126914621947431253958760284636283 13
24077 : 372755046202406864831800655564258898513839322780658959041340629486059044508 33 : 1)
Mensagem decifrada: mensagem60
Decifração bem-sucedida! A mensagem decifrada corresponde à original.

Número da mensagem: 13
Índice no vetor ciphertext_vector: 65
Chave privada (sk): 331563995889166225180744508819973039765280777193610788011025468153559727
6793
Chave pública (pk): (416651095429591066630081866566726683001016215309631988652608298874437 65
231462 : 418248846044922991822957728519478218588569458790989671585080654918473333201 49 : 1)
Mensagem decifrada: mensagem65
Decifração bem-sucedida! A mensagem decifrada corresponde à original.

Número da mensagem: 14
Índice no vetor ciphertext_vector: 67
Chave privada (sk): 102765819267066341075336548683369597267351872134591479721061953794760656
5680
Chave pública (pk): (158088509625244430165443196723802573357055889274457154779604385615177 18
009674 : 368153913810096266724475114741272227001644033759930111173880470282435 39723 59 : 1)
Mensagem decifrada: mensagem67
Decifração bem-sucedida! A mensagem decifrada corresponde à original.

Número da mensagem: 15
Índice no vetor ciphertext_vector: 75
Chave privada (sk): 716686550450573918346800966016220293649531851042483572298074232577139125
307
```

Chave pública (pk): (511505830789672423031378083416972411772561976784581323825171444075931149104400 : 2930155553861223886730242318746913314937652801106059852900004227380500541897810 : 1)
Mensagem decifrada: mensagem75
Decifração bem-sucedida! A mensagem decifrada corresponde à original.

Número da mensagem: 16
Índice no vetor ciphertext_vector: 80
Chave privada (sk): 1048490973091317978796478015059604346749682215021246499872792729583210001089
Chave pública (pk): (312292168347307444548296271143764345456003621563390968573372781217172467112331 : 269782549331547643299537674326119125865862339006717303767263685241517597473910 : 1)
Mensagem decifrada: mensagem80
Decifração bem-sucedida! A mensagem decifrada corresponde à original.

Número da mensagem: 17
Índice no vetor ciphertext_vector: 83
Chave privada (sk): 6350216855687878001040131685010078857539274565696697982962597774583431582730
Chave pública (pk): (1303771095526366469233503340088705564849935962572968357039923633734038611023 1 : 3899181667954708555111836532880781421096859743927172385492749778803814038725210 : 1)
Mensagem decifrada: mensagem83
Decifração bem-sucedida! A mensagem decifrada corresponde à original.

Número da mensagem: 18
Índice no vetor ciphertext_vector: 88
Chave privada (sk): 4980495920358281932762681270230493448640330371376986797529661943300307779306
Chave pública (pk): (1973046898494322236767231845896022407531411632742794518945922418197474543609 5 : 879950471740274931831642454236952833722796220841738336077848470763590622497810 : 1)
Mensagem decifrada: mensagem88
Decifração bem-sucedida! A mensagem decifrada corresponde à original.

Número da mensagem: 19
Índice no vetor ciphertext_vector: 91
Chave privada (sk): 6520211552441537830684040298599474565668504421717213456864422719898098125154
Chave pública (pk): (5214975726001668596089449117978367758205746008450996214557960604210431901411 7 : 1799006458445989435740859711999976494828454408258811984272385098817893238977 9 : 1)
Mensagem decifrada: mensagem91
Decifração bem-sucedida! A mensagem decifrada corresponde à original.

Número da mensagem: 20
Índice no vetor ciphertext_vector: 92
Chave privada (sk): 2569255557812050470171165969906750627168869141463718443178725704897973537688
Chave pública (pk): (4079308141877894113223334243732638831827033737240351531150226366279706172089 9 : 2826797548191448835589240835154090050807337537557283511384100749148936970490 9 : 1)
Mensagem decifrada: mensagem92
Decifração bem-sucedida! A mensagem decifrada corresponde à original.