Enunciado

Este trabalho usa SageMath nas suas implementações

1. Pretende-se construir em torno de uma cifra assimétrica um conjunto de técnicas criptográficas destinadas a fins distintos. Apesar de todas as alíneas do problema poderem ser respondidas com a maioria das cifras assimétricas clássicas ou pós-quânticas, neste problema vamos exemplificar o processo com uma técnica simples da família Diffie-Hellman nomeadamente a cifra assimétrica ElGamal com parâmetros de segurança λ.
   A. Implemente um esquema PKE $\mathsf{ElGamal}(\lambda)$ (ver Capítulo 4) num subgrupo de ordem prima q, com $|q| \geq \lambda$, do grupo multiplicativo $\mathbb{F}_p^*$ com p um primo que verifica $|p| \geq \lambda \times |\lambda|$. Identifique o gerador de chaves e os algoritmos de cifra de decifra neste esquema. Identifique o núcleo deterministico do algoritmo de cifra.
   B. Supondo que a cifra que implementou é IND-CPA segura (de novo Capítulo 4), usando a transformação de Fujisaki-Okamoto implemente um PKE que seja IND-CCA seguro.
   C. A partir de (b) construa um esquema de KEM que seja IND-CCA seguro.
   D. A partir de (b) construa uma implementação de um protocolo autenticado de "Oblivious Transfer" κ-out-of-n.

Exercício 1.a.

EL GAMAL Qualquer PKE é determinado por três algoritmos: geração de chaves, cifra e decifra:

$\mathrm{GenKeys}(\lambda)$ ... λ é o parâmetro de segurança

- gerar aleatoriamente um primo $q \approx 2^\lambda$
- gerar um primo p tal que $\mathbb{F}_p^*$ tem um sub-grupo de ordem q ; calcular um gerador g desse sub-grupo
- gerar aleatoriamente $0 < s < q$, a chave privada
- calcular e revelar a chave pública $\mathsf{pk} \equiv \langle p, q, g, g^s \rangle$

$\mathrm{Enc}(\mathsf{pk}, m)$ ... a mensagem m é um elemento de $\mathbb{F}_p^*$

- obter elementos públicos $p, q, g, g^s \leftarrow \mathsf{pk}$
- gerar aleatoriamente $0 < \omega < q$
- calcular $\gamma \leftarrow g^\omega$ e $\kappa \leftarrow (g^s)^\omega$.
- construir o criptograma $\mathbf{c} \leftarrow \langle\, \gamma\,,\, m \times \kappa \,\rangle$

Note-se que se verifica $\kappa = \gamma^s$.

$\mathrm{Dec}(\mathsf{sk}, \mathbf{c})$ ... $\mathsf{sk} = s$ é a chave privada

- obter a chave privada s
- obter o criptograma $\mathbf{c} = \langle \gamma, \delta \rangle$
- calcular $\kappa \leftarrow \gamma^s \mod p$
- calcular $\kappa^{-1} \mod p$
- recuperar a mensagem original: $m \leftarrow \delta \times \kappa^{-1} \mod p$

In [1]:
```
print_tentativas_genKeys = False
```

In [2]:
```
from sage.all import *
lambda_security = 128  # Define um tamanho de bits para q
```

Para encontrar um gerador do grupo multiplicativo temos que encontrar um número cuja ordem seja igual à do grupo, i.e.:

- Dada a ordem do grupo multiplicativo $F_p^* = \phi(p)$ o gerador g tem de ter ordem $\phi(p)$, ou seja, $g^{\phi(n)} = 1 \; mod \; p$

In [3]:
```
def find_generator(p):
    """Encontra um gerador do grupo multiplicativo F_p^*."""
    if not is_prime(p):
        raise Exception("O p de input não é primo")

    phi_p = p - 1  # Para p primo, phi(p) = p - 1

    fatoracao = factor(phi_p)
    fatores_primos = list(set([q for q, e in fatoracao]))

    #print(f"Fatoração de phi(p): {fatoracao}")
    #print(f"Fatores primos de phi(p): {fatores_primos}")

    # Itera sobre possíveis geradores
    for g in range(2, p):
        if gcd(g, p) != 1:
            continue  # Ignora números não coprimos com p

        is_gerador = True
        for q in fatores_primos:
            if pow(g, phi_p // q, p) == 1:
                is_gerador = False
                break

        if is_gerador:
            return g

    return None
```

In [4]:
```
def gen_keys(lambda_security):
    print("---------------------------------Geração de chaves:----------------------------
    # Gerar aleatoriamente q com bit_length maior que lambda
    q = random_prime(2^129 - 1, False, 2^128)
    print("Parâmetro q gerado:",q)

    #Gera-se sucessivamente inteiros pi = q*2^i+1 até que pi seja um primo suficientemente
    i = 1
    p_i = q * (2^i) + 1
    tamanho_desejado = lambda_security * lambda_security.nbits()

    while True:
        # Caso o p_i não convergir tenta um novo q
```

```
        if p_i.nbits() > 2500:
            q = random_prime(2^129 - 1, False, 2^128)
            i = 1
            p_i = q * (2^i) + 1
            print("p não convergiu, novo valor de q: ",q)

        if is_prime(p_i) and p_i.nbits() >= tamanho_desejado:
            break

        i += 1
        p_i = q * (2^i) + 1
        if print_tentativas_genKeys:
            print(f"Tentativa {i}: p_i = {p_i} (Tamanho: {p_i.nbits()} bits)")

    p = p_i
    print("Parâmetro p gerado:",p)

    # Fp é um grupo multiplicativo se para todo o x pertencente a Fp gcd(x,p) = 1 (trivial
    # Criar o corpo finito F_p
    F_p = GF(p)
    # O grupo multiplicativo F_p^* é o conjunto de elementos não nulos de F_p
    F_p_star = F_p.unit_group()

    # Agora obtemos um gerador do subgrupo de ordem q
    g = find_generator(p)
    print("gerador (g):",g)

    # Gerar aleatoriamente a chave privada 0<s<q
    s = randint(1, q-1)
    print("chave privada (s):", s)
    print("----------------------------------------------------------------------------------
    return (p, q, g, pow(g, s, p)), s
```

Escolha de p:

A ordem desse grupo é $n = p - 1$ e para que o DLP seja complexo não basta apenas que p seja grande: é também necessário que o maior factor primo de $(p-1)$ seja também grande. Para garantir estas condições o primo p é gerado de uma determinada forma:

1. Gera-se um primo q grande: com mais de λ bits de tamanho; este vai ser o maior factor de $(p-1)$
2. Gera-se sucessivamente inteiros $p_i = q\,2^i + 1$ até que p_i seja um primo suficientemente grande .

Como é que F_p^* tem um subgrupo de ordem q? (Teorema de Lagrange)

O Teorema de Lagrange diz que, se um grupo G tem ordem finita e H é um subgrupo de G, então |H| é um divisor de |G|.

Visto que F_p^* tem ordem $p-1$ e $p-1$ divide $2q$ (e consequentemente q) então podemos afirmar que F_p^* tem um subgrupo de ordem q.

In [5]:

```
def enc(pk, m):
    p, q, g, g_s = pk
    omega = randint(1, q-1)  # Escolhe um valor aleatório ω entre 0 e q
```

```
    gamma = pow(g, omega, p)  # γ = g^ω mod p
    kappa = pow(g_s, omega, p)  # κ = (g^s)^ω mod p
    c = (gamma, (m * kappa) % p)
    return c
```

In [6]:
```
def dec(sk, pk, c):
    p, _, _, _ = pk
    gamma, delta = c
    kappa = pow(gamma, sk, p)  # κ = γ^s mod p
    kappa_inv = inverse_mod(Integer(kappa), Integer(p))  # Calcula o inverso de κ mod p
    m = (delta * kappa_inv) % p  # Recupera a mensagem original
    return m
```

In [7]:
```
# Exemplo
pk, sk = gen_keys(lambda_security)  # Geração de chaves
m = randint(1, pk[0]-1)  # Mensagem aleatória em F_p*
c = enc(pk, m)  # Cifra a mensagem
m_dec = dec(sk, pk, c)  # Decifra a mensagem

# Exibir os resultados
print(f"Mensagem original: {m}")
print(f"Criptograma: {c}")
print(f"Mensagem decifrada: {m_dec}")
print(f"Decifração bem-sucedida? {m == m_dec}")
print("--------------------------------------------------------------------------------")
```

```
----------------------------------Geração de chaves:-------------------------------------
Parâmetro q gerado: 48769969887673781855248751630823219787 9
Parâmetro p gerado: 150621823104779967952674486310980997068950933002572273556298979376990823
322843362521548350114621631119750833468936656890691575677553352150531335201524919595330229488
885445506368100768277663872131874863667510902448081982633952994312967279522821334078315252747
973535471383940865746674369375861170481461652444521938571933320006321838212011723650892600 5
1393150977
gerador (g): 3
chave privada (s): 352014016105149690688414936672900546621
--------------------------------------------------------------------------------
Mensagem original: 993242793758555459565595555937848523279772650440560007467842262428857150 2
784179366694889588272588459272946929591977903419159642944298895271470183786461689655996711421
044824772529335169561777376398392997461690707476183329872993776434680182850108075042933815 8
720713535873612311062882884236153524522179553095569991613374184113028859042204778343598667 17
72898247
Criptograma: (208301975061275491494087413806577418796907478673655592585267144866552623173310
192399557543113415082130895298332121094820140208976499883837269426842928416200117205679062 55
258878829517386376726596327102438427728030555186824319757989453173539144921583432246724088 75
644738521407612429422016500242093045775845418020325897522123587178389718636946568990187929 84
599, 114232149498140636439714417591007041550592306527267980361691433491237634878125229982040
599034601265717894223880433044578048710947202448270870469626760000637703544797199645044053 29
592063585659356738765026457953981277853899148029824172064698506711122096614815875092404178 20
903713005002526564850643284450140583554594293046772748360244552768052479922482095216572)
Mensagem decifrada: 993242793758555459565595555937848523279772650440560007467842262428857150
278417936669488958827258845927294692959197790341915964294429889527147018378646168965599671 14
210448247725293351695617773763983929974616907074761833298729937764346801828501080750429338 15
872071353587361231106288288423615352452217955309556999161337418411302885904220477834359866 71
772898247
Decifração bem-sucedida? True
--------------------------------------------------------------------------------
```

Exercício 1.b.

Supondo que a cifra que implementou é IND-CPA segura (de novo Capítulo 4), usando a transformação de Fujisaki-Okamoto implemente um PKE que seja IND-CCA seguro.

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.

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

Definições de variáveis

In [8]:
```
print("---------------------------------Definição de veriáveis----------------------------
lambda_bits = 128

pk, sk = gen_keys(lambda_bits)
x = randint(1, pk[0]-1)  # Mensagem aleatória em F_p*
length_in_bytes = (x.bit_length() + 7) // 8
x = x.to_bytes(length_in_bytes, byteorder='big')
print("length x:",len(x))
print("Mensagem de input:",x)
print("-----------------------------------------------------------------------------------
```

```
---------------------------------Definição de veriáveis---------------------------------
---------------------------------Geração de chaves:-------------------------------------
Parâmetro q gerado: 67826770315088323833199257602029405976 3
Parâmetro p gerado: 284828789229163771187790577473635873555798367063503005037831047771526739
928035601633062054324988241820076963902308682872861791497645842452824668204470992375144337 69
195228945701378669088294907091705668419869643359852322460539023150626905065599215911071394 56
570073778561963488303940167514760390137894168715976740240956882581016293943439678400270703 78
054817390763149146847310892382957568358932691385679241501267331234017342063506721490377025 81
409165811820632400566155068253653115317868809400891739887742963506927817703883403780861434 28
24711684097
gerador (g): 3
chave privada (s): 53194977355562509103639693686620898397 0
--------------------------------------------------------------------------------
length x: 226
Mensagem de input: b"\x02\x99\x9f\xfa\x8c\xa5MH\x1d\xae&\xb7\x1a\xc1\xbd\xe5H\x1f\xbf'\xaa8
\xf0\x97\x17\x04\x96E\x07\xb43\x03\xdb0i\x808\xb7\xf8\xa2@\x0e@\xfdN\xcd\xde?\xc2\x9a!\x969
\xf6\xdaeVr7\xad\x08\xb3\xf4\x16\xdf\x9b\xfa#\xa4#\xd2\xb3\xa4\xaf\x9b\x06~,a55\xbb\xb2!\xca
\x888TT\xdc\xbb%\xdf3b\xbc\xb2\x81A]\x92\x8c\xa2\xeel\x15\xa5~Z\xe5UB9l\xba8#\xac\xc8\xa0\x9
a\xd39t\xcf\xf5\xb26\xe4S\x9f\xa9\xa2z\xee\xf8\xae\xd8\xd5\xe0\xda\x10\xf7EE\xf6\xd0\xca\xfd
\xbbN\x1a(\x93\xebN\x17\x06\x1a\xa0\xda\xf8\x8e.\xba\x1b\xc6\x1d<$\x01\xccv\x02O\x17#;R\x9e
\xb1r\xe6\xc7\xa2\xebi\xb4\x1f\xb5\x14\x99\xd1\xf9\xca\xd9jN\x92J\x1a\x08d4\xbd\x98/\xc9\x8f
\xe6\xd5\xd8?23\xc2\xfdr%s6\xf6\x0bK7\xad"
--------------------------------------------------------------------------------
```

Funções auxiliares:

- g, um "hash" pseudo-aleatório de tamanho $|x|$
- h, um "hash" pseudo-aleatório de tamanho $lambda$
- f_p o núcleo determinístico da cifra ElGamal

In [9]:
```
import hashlib

def g(r):
    """Hash pseudoaleatório g(r) com tamanho igual ao da mensagem x"""
    g = hashlib.sha512()
    r_bytes = r.to_bytes((r.bit_length() + 7) // 8, 'big')
    g.update(r_bytes)
    final_hash = g.digest()  # Truncar para o tamanho de x
    while len(final_hash) < len(x):
        g = hashlib.sha512()
        g.update(r_bytes)
        final_hash += g.digest()
    #print("g(r) hash:", final_hash[:len(x)])
    return final_hash[:len(x)]

def h(r, y):
    """Hash pseudoaleatório h(r, y) com tamanho lambda_bits"""
    h = hashlib.sha512()
    r_bytes = r.to_bytes((r.bit_length() + 7) // 8, 'big')
    ry = bytes(a ^^ b for a, b in zip(r_bytes, y))
    h.update(ry)
    full_hash = h.digest()[:lambda_bits // 8]  # Truncar para lambda bits
    #print("h(r, y) hash:", full_hash)
    return full_hash

#Núcleo determinístico da função enc anterior
def f_p(pk, r, rlinha):
    p, q, g, g_s = pk
    gamma = pow(g, rlinha, p)  # γ = g^ω mod p
    kappa = pow(g_s, rlinha, p)  # κ = (g^s)^ω mod p
    return (gamma, (r * kappa) % p)
```

Cifra IND-CPA segura

In [10]:
```
print("---------------------------------CIFRA---------------------------------")
def enc_fujisaki(pk,x):
    r = ZZ.random_element(2^(lambda_bits - 1), 2^lambda_bits)
    print("r:",r)
    y = bytes(a ^^ b for a, b in zip(x, g(r)))
    rlinha = h(r,y)
    c = f_p(pk,r,int.from_bytes(rlinha,"big"))
    return (y,c)

(y,c) = enc_fujisaki(pk,x)
print("(y,c) = ",(y,c))
print("-----------------------------------------------------------------------------
```

```
----------------------------------CIFRA---------------------------------
r: 23805011132251930299953506989971433488 6
(y,c) =  (b'u\xfa\xe4\t\xca\t\x0c\xdb\x1dv\xf8-b\x12\x87\xaec\xf5_+r\xa5I\x84n\xe3\xbb^x\x1e
\x11+\x85C\xe9\xd8\x8f\x01\xed\xd8\xb0\xcc\xb03\x8aJ=S\x04\xdb\xe0%\xa9L \x07?\xa1\x19\x17\x
04\xfeiF\xa8\xf8\x81\xd0\xe2\x8f\x93 \xa4wE\x9c\x06\xff[~\x1eQR-\x12\x15\x81G-;\x96>\xa0\x99
@\x94\xec\xf2\xc1\x05%:\xb7\x94\x9c\xd7U\xb0\x9eb\xb6.\xff-{\x8b\xb3\x162\xc2\xf3\x00\x17\xc
e\xc3\xb8/f\x930\xe4Z\xe4\xd6\xafk\xae\x00\x0bz\xa2\xc3\xcd\x0en\x1c0\xc6%&\xf7\tQt\xc6Uh\xa
c8\x88\x84\x8b\x0ev\r\xad\xd3g\xcc\xe6\xf1\x02\xb2\x85\xac{\xe5z\x93-!\xc8\x1c\xa5\xcb8G\x0e
\x13\xf8\x89\xc9\xa6\x9a\xb1*,\xe2\xd3\xd9\x1a\xd0\xba\xae\xc5K\x15\x82\xa4\x0c5\xd4\xe7\xaf
\x8a\xd1\x84\x95\x08hI\\)ci\xde', (55157584727068968557270277751052837805742751203797466793141992339464306859292477029283210496614797257633169876528824121788645479658383201221359500803704577599110938543039717556458868785126127117125286781446965879390093150325098460921784216230161091152992995401481291644173687087555324052547876651122325952464878743206745119832090924642375689392456963225337652467917247158892615366381352333528601879988236799403751343936174826467241250617685754014429183439022574901900387221989840734527492044911817271085847741259060250490616647511081147787678, 45418896692305408580065643067037996796916331050155298217499559731440528986722762216838509257331966172755785031914251969138788699798363393168194459517588669055011068538668732095267165621360775895835307571661005958887855290329721606674237671090095588163043503146634148295126534478944419000417347812887609781290384087036987758523462049083395394209649230702194516799291349400704429727974917481421287959862587884902603290850761349228122229251331449042450900060601829308744632600560851178461855530127925304302633289246040776037296685179286903620102))
-----------------------------------------------------------------------------------------------
```

Decifra

In [11]:
```
print("---------------------------------------DECIFRA---------------------------------------")
def dec_fujisaki(sk,pk,y,c):
    r = dec(sk,pk,c)
    print("r:",r)
    rlinha = h(r,y)
    if c != f_p(pk,r,int.from_bytes(rlinha,"big")):
        raise Exception("O criptograma não corresponde a fp(r,r'), absurdo")
    else:
        res = bytes(a ^^ b for a, b in zip(y, g(r)))
        return res

print("Mensagem decifrada: ",dec_fujisaki(sk,pk,y,c))
print("x == dec_fujisaki(sk,pk,y,c)?: ",x == dec_fujisaki(sk,pk,y,c))
print("-------------------------------------------------------------------------------------")
```

```
---------------------------------------DECIFRA---------------------------------------
r: 23805011132251930299953506989971433488 6
Mensagem decifrada:  b"\x02\x99\x9f\xfa\x8c\xa5MH\x1d\xae&\xb7\x1a\xc1\xbd\xe5H\x1f\xbf'\xaa
8\xf0\x97\x17\x04\x96E\x07\xb43\x03\xdb0i\x808\xb7\xf8\xa2@\x0e@\xfdN\xcd\xde?\xc2\x9a!\x969
\xf6\xdaeVr7\xad\x08\xb3\xf4\x16\xdf\x9b\xfa#\xa4#\xd2\xb3\xa4\xaf\x9b\x06~,a55\xbb\xb2!\xca
\x888TT\xdc\xbb%\xdf3b\xbc\xb2\x81A]\x92\x8c\xa2\xeel\x15\xa5~Z\xe5UB9l\xba8#\xac\xc8\xa0\x9
a\xd39t\xcf\xf5\xb26\xe4S\x9f\xa9\xa2z\xee\xf8\xae\xd8\xd5\xe0\xda\x10\xf7EE\xf6\xd0\xca\xfd
\xbbN\x1a(\x93\xebN\x17\x06\x1a\xa0\xda\xf8\x8e.\xba\x1b\xc6\x1d<$\x01\xccv\x02O\x17#;R\x9e
\xb1r\xe6\xc7\xa2\xebi\xb4\x1f\xb5\x14\x99\xd1\xf9\xca\xd9jN\x92J\x1a\x08d4\xbd\x98/\xc9\x8f
\xe6\xd5\xd8?23\xc2\xfdr%s6\xf6\x0bK7\xad"
r: 23805011132251930299953506989971433488 6
x == dec_fujisaki(sk,pk,y,c)?:  True
-----------------------------------------------------------------------------------------------
```

Exercício 1.c.

Associado a um mecanismo de encapsulamento de chaves existe um algoritmo de revelação ou KRev (“*key revelation*”), que a partir do encapsulamento de uma chave, revela-a. Naturalmente, por analogia com as cifras assimétricas, o KEM é um algoritmo público, total e probabilístico, enquanto que o KRev é um algoritmo privado, parcial e determinístico.

A condição de correção será

$$(e,k) \leftarrow \mathsf{KEM} \quad \text{sse} \quad \mathsf{KRev}(e) \simeq k$$

Combinando o exercício anterior com um KEM o esquema pode-se descrever pelo par de algoritmos:

$$\begin{cases} E(x) & \equiv & \vartheta\,(e,k) \leftarrow \mathsf{KEM}\ .\ (e\,,\ k \oplus x) \\ D(e,c) & \equiv & \vartheta\,k \leftarrow \mathsf{KRev}(e)\ .\ k \oplus c \end{cases}$$

Substituindo KEM pela função de cifra do exercício (b) obtemos uma segurança IND-CCA para a encapsulação da chave. Como o KEM é um mecanismo apenas de encapsulamento da chave iremos utilizar em conjunto com um DEM (neste caso OTP) para demonstrar que resulta para cifrar uma mensagem, como é demonstrado no diagrama:

$$\begin{array}{ccccc} \mathrm{KEM} & \rightarrow & \mathrm{encapsulation} & \rightarrow & \mathrm{KRev} \\ \downarrow \mathrm{key} & & & & \downarrow \mathrm{key} \\ \oplus & \rightarrow & \mathrm{Ciphertext} & \rightarrow & \oplus \\ \uparrow & & & & \downarrow \\ \mathrm{plaintext} & & & & \mathrm{plaintext} \end{array}$$

Definições de variáveis

In [12]:
```
lambda_bits = 128

pk, sk = gen_keys(lambda_bits)
x = randint(1, pk[0]-1)  # Mensagem aleatória em F_p*
length_in_bytes = (x.bit_length() + 7) // 8
x = x.to_bytes(length_in_bytes, byteorder='big')

print("Mensagem aleatória em F_p*: ",x)
```

```
----------------------------------Geração de chaves:-------------------------------------
Parâmetro q gerado: 37996796567432510776340227867370455777 9
Parâmetro p gerado: 30434054353124265534944869285186846262805282951588429607865665856706534254111402813636095612020141401626077098230011061332427021288873458542774734276014405518161496750840673683827805529014079721760893031178039716746559291350329308533710470422174799097842161989583517907477631241455990493586215410963044483710233245270783600697967701731511459492761690338458010390620160012142412212112720347120465324900233473446673785221764712575937200890082980742253664921968285477922582265969922255233493154684177285019937782537254946557478167205484953 7
gerador (g): 3
chave privada (s): 291725272953963459071538182785024684384
-----------------------------------------------------------------------------
Mensagem aleatória em F_p*:  b'7\xb6\x128\xf9j\x06\xecb\\?\x11\xcd\xb2\xc8E\xd43E\xba5~\xb6\x80B\xbb\x05\x86N\x97L\xa8|\x8a\x1f\x89\x8av]!\xb7\x93\xea\xb8\xcbk\xad@Y\xe5\xee\xc4\x0b\xaa\xc3I\xde\x02\xa8\xd5\xf1\x8fC>)\xe2\xc9\xcf\xfe\xda\x0b\x11]zO\xe6\xc2<\xe5.\x90\x05\xea\xcdt2\x18ga:\xd4A\xe1\x05\xe2^><\xe7\xc4\xeeQJ\x90=\xe93E\x8f0\x8e\x1cDV\xe5\x06D\xe8\x1eb-g\x18vUpDjp\xdd\xe7R\xcc\xd23<\x16\x9c\x8cy\x80L\xaaN\xc4\xd6\xcb\x82)\xfa\x848c)Fun\xcf\xb2\x87I\x03\xda\x17\x12\xe5\x7f\xfc\xa6u\xbcn\xd5\xa8\t\xad\x1d\x16\xf2\x80\x82\x13\x81\x947\xe7\x1c\xdb\x0bE\xe2~\x8dw,\x89\xa7S_%#\xbd\x81Z\x9c\x02\x80\x98\xb8i\xfa\x8b\x19\xdah\xea\x01cNE\x9a\x93['
```

KEM + DEM Encrypt

Construir algoritmo de cifra tal que $E(x) \equiv \vartheta(e,k) \leftarrow KEM \cdot (e, k \oplus x)$ onde

- k é gerado aleatoriamente;
- e é o encapsulamento dessa chave.

In [13]:
```
def kem_encrypt(pk, x):
    # Gerar chave simétrica aleatória k
    k = ZZ.random_element(2^(8*len(x)-1), 2^(8*len(x))).to_bytes(len(x), 'big')
    print("k: ",k)
    # Utilizar cifra do ex1.b. como KEM para encapsular k
    e = enc_fujisaki(pk,k)

    # Aplicar DEM (neste caso OTP) da chave gerada k com a mensagem x
    ciphertext = dem(x,k)

    return e, ciphertext

def dem(x,k):
    ciphertext = bytes(a ^^ b for a, b in zip(x, k))
    while len(ciphertext) < len(x):
        ciphertext += bytes(a ^^ b for a, b in zip(x[len(ciphertext):], k))
        ciphertext[:len(x)]
    return ciphertext
```

KEM + DEM Decrypt

A decifração é dada por: $D(e,c) \equiv \vartheta k \leftarrow KRev(e) \cdot k \oplus c$ onde

- $KRev$ no nosso caso é a técnica de decifração utilizada no exercício 1.b.;
- e é o encpasulamento da chave gerada;
- c o criptograma recebido de KEM.

In [14]:
```
def kem_dem_decrypt(sk, pk, e, ciphertext):
    (y, c) = e
    k = dec_fujisaki(sk, pk, y, c)
    print("k: ",k)
    x = dem(ciphertext,k)
    return x
```

Teste

In [15]:
```
print("-------------------------------------------------------------------------------")
print("mensagem (x): ",x)
e, ciphertext = kem_encrypt(pk,x)
print("-------------------------------------------------------------------------------")
print("e: ",e)
print("-------------------------------------------------------------------------------")
print("k (+) x: ",ciphertext)
print("-------------------------------------------------------------------------------")

mensagem = kem_dem_decrypt(sk, pk, e, ciphertext)

print("mensagem (final): ",mensagem)
print("-------------------------------------------------------------------------------")
print("mensagem inicial == mensagem final ?: ",x == mensagem)
print("-------------------------------------------------------------------------------")
```

```
------------------------------------------------------------------------------------------------
-----------------------
mensagem (x):  b'7\xb6\x128\xf9j\x06\xecb\\?\x11\xcd\xb2\xc8E\xd43E\xba5~\xb6\x80B\xbb\x05\x
86N\x97L\xa8|\x8a\x1f\x89\x8av]!\xb7\x93\xea\xb8\xcbk\xad@Y\xe5\xee\xc4\x0b\xaa\xc3I\xde\x02
\xa8\xd5\xf1\x8fC>)\xe2\xc9\xcf\xfe\xda\x0b\x11]zO\xe6\xc2<\xe5.\x90\x05\xea\xcdt2\x18ga:\xd
4A\xe1\x05\xe2^><\xe7\xc4\xeeQJ\x90=\xe93E\x8f0\x8e\x1cDV\xe5\x06D\xe8\x1eb-g\x18vUpDjp\xdd
\xe7R\xcc\xd23<\x16\x9c\x8cy\x80L\xaaN\xc4\xd6\xcb\x82)\xfa\x848c)Fun\xcf\xb2\x87I\x03\xda\x
17\x12\xe5\x7f\xfc\xa6u\xbcn\xd5\xa8\t\xad\x1d\x16\xf2\x80\x82\x13\x81\x947\xe7\x1c\xdb\x0bE
\xe2~\x8dw,\x89\xa7S_%#\xbd\x81Z\x9c\x02\x80\x98\xb8i\xfa\x8b\x19\xdah\xea\x01cNE\x9a\x93['
k:  b'\x8bC\xa4\xe4\xe7R\xc3\xa8~\xa2\xb2%\x9a\x1aP\x88j+4,1r\x8b5\xd9\t`\x9fqh\xad\x86\xb63
\x96\x0e\xf8\xeaA\xb9\xb6\x14UI\xd9\xcc\xe2 \xd3\x8au?j-6>s\x12H8\xf6tP\x0bdwT\x18\xdaR\x86
\x9f\x10\x1a\xee\t\xd4/J/\x13c\x89f\xef(\xc0\x85$/\x95i\x80\x0c\xdd\xbd\x1f\xba\x90\x985%\xb
4h\x9ay)n\x0b\xdb\x80\xc4\xee{CK\xc0"\x91N\x8a=w\x04\x9d4\x96\xbcL\x93\x8e\xc9\x8d\x8d\xe1;
\x132\xdfP\xdc5\x831\xef\xf1\x80/[\xd5\x19k\x9e\xae\x840\xca\xabK\xfe\xc85\x90\x00\xe0\xce\r
\xb1\x8fQ\xef\xa0\xb54\xba\xee\xeae\xd4\x90\xdb{\x90\x8cK\x1e\xd5r&\x95k\xba\xfc-q\xd8\r\x04
ZJ\xd1\x18\x96\xeb\x0e\x8c\x90\r\x9c\x8aiX\xb6\x97\xde\x10\xd9W)\x91\xca_\x8d'
r: 29699731575897517827835094456493482172 7
------------------------------------------------------------------------------------------------
-----------------------
e:  (b"\xa1\xc4UC\x90p\xc8\x1b\x1f\x89\x0f\xdaS\xb19\xf3\x81\xdf+\xcc\xfb;\xb6b\x11\xdeJ\xde
dU\x05\xa3\xdf\x1fm\x88\xbc\x10\x81Q\x9f2\xd2\xdb\x7f\xf0Zw\xbe\xb4\xa4d\xaeE\x017C\xdc\xa3D
\x92\x86\xd2\xc1N\xf0\xa5\xbf\xadp\x8d,q1S\xf6\x1d\x84#T\xf8\x97\x96\x86%a\xfd\xd2\xec\xf8\x
bf(\x951u\x98v\x96k\x1eq\xdft\x80\xb3_\xae\xfc\xad\xe78\x93\x83E\x92\x10\x04J\xa6G\xba\xf3\x
9cx\xf9\xc6\x14vf\x14\x7fn\xfa\xaf\xea\x88r\x19b\xaf\x15\x9e\xeaJ\x04\x05\x9f\xcf\x91\x9c$<V
y\xaeq\xdf\x96\xe3\xdb\xa1\x19k\x86\xa44\xcdY\xa6wh2\x13\x08\x02\xb9\x87[\x05\xcb\x1f\x13\xa
7\x85{\xd0>\x0eBg\xe9p\xd6\xaa\x80\x7fz&Q\xf9\xb03+\x14\xc7'\xf9vw~v\xb8|\xde\xe3G\x11\x80\x
03\xd0\xdfb%", (85702133062264776292396264379094898978799602575494607562001806336871607461893
64440972815891408837556672484560783416243802431188624688854631921301058143274367008119138928
31418488662504503662114126743114240471030370268212822349458963342024691820889245595397330004
27903235444091870140874629332990515714245962776066951855229408351936101819170442443039770147
29823476663603696116589038252182962771140941766573630358704807588549667225110271663770249867
36373055124909323234610001555285503652300811700101513665143845025767775169619682379218288989
9, 18654406596259733229388056819274311857432025571884730629937020698674408261531050783435873
49694062589558822376273219155479223842162123446686695544359376465153646907815858820190067479
70639228137915321058181388232824114103364785443552361272308964516199281330167668387313769959
40898622226219940510838103204818053395683958491294079405457263957162613036459590629582941522
35379481862947487551604411503733247154505413022519114493317371622615398137597244633409977552
48646568246292525973239370072352991022989682976610412171834119637760346326211427))
------------------------------------------------------------------------------------------------
-----------------------
k (+) x:  b"\xbc\xf5\xb6\xdc\x1e8\xc5D\x1c\xfe\x8d4W\xa8\x98\xcd\xbe\x18q\x96\x04\x0c=\xb5\x
9b\xb2e\x19?\xff\xe1.\xca\xb9\x89\x87r\x9c\x1c\x98\x01\x87\xbf\xf1\x12\xa7O`\x8ao\x9b\xfba\x
87\xf5w\xad\x10\xe0\xed\x07\xfb\x135M\x95\x9d\xd7$\x88\x8d\x8eM`\xa1\xef\x16\x13\xaf\x01\x83
fc\xab\x9b\x1a\xd8\xe2E\x15A(a\t?\xe3!\x86w\\\xdbt\xfe\xf8\xa7\x90\x1a+\x84\xeb\x0e\xd8\xaa-
\xa6M\x84\xca\x8f,\xa7Zor\xc8D\xd2\xd6<Ni\x9bA_\xd2\x07\x05\xaeS)\\y)\x7f+'K\xadr/\x9dS\xfd
\x87\xc2E\xa4d\xf9y\x816J\x17\xf2+rM)$S\xce`\x9c\xb3C\xf7s&\x10Yh\x11\x18|\xf9\xc9\xa9-\xd0
\x89\xc4qZ]Q\xaaW\x05o\xf2\xa5\x17\xb1\x92\x8e\x10\x95$\xe3\x93\xd3\xafM\xb6\xfa\xd84g\xd4P
\xcc\xd6"
------------------------------------------------------------------------------------------------
-----------------------
r: 29699731575897517827835094456493482172 7
k:  b'\x8bC\xa4\xe4\xe7R\xc3\xa8~\xa2\xb2%\x9a\x1aP\x88j+4,1r\x8b5\xd9\t`\x9fqh\xad\x86\xb63
\x96\x0e\xf8\xeaA\xb9\xb6\x14UI\xd9\xcc\xe2 \xd3\x8au?j-6>s\x12H8\xf6tP\x0bdwT\x18\xdaR\x86
\x9f\x10\x1a\xee\t\xd4/J/\x13c\x89f\xef(\xc0\x85$/\x95i\x80\x0c\xdd\xbd\x1f\xba\x90\x985%\xb
4h\x9ay)n\x0b\xdb\x80\xc4\xee{CK\xc0"\x91N\x8a=w\x04\x9d4\x96\xbcL\x93\x8e\xc9\x8d\x8d\xe1;
\x132\xdfP\xdc5\x831\xef\xf1\x80/[\xd5\x19k\x9e\xae\x840\xca\xabK\xfe\xc85\x90\x00\xe0\xce\r
\xb1\x8fQ\xef\xa0\xb54\xba\xee\xeae\xd4\x90\xdb{\x90\x8cK\x1e\xd5r&\x95k\xba\xfc-q\xd8\r\x04
ZJ\xd1\x18\x96\xeb\x0e\x8c\x90\r\x9c\x8aiX\xb6\x97\xde\x10\xd9W)\x91\xca_\x8d'
mensagem (final):  b'7\xb6\x128\xf9j\x06\xecb\\?\x11\xcd\xb2\xc8E\xd43E\xba5~\xb6\x80B\xbb\x
05\x86N\x97L\xa8|\x8a\x1f\x89\x8av]!\xb7\x93\xea\xb8\xcbk\xad@Y\xe5\xee\xc4\x0b\xaa\xc3I\xde
\x02\xa8\xd5\xf1\x8fC>)\xe2\xc9\xcf\xfe\xda\x0b\x11]zO\xe6\xc2<\xe5.\x90\x05\xea\xcdt2\x18g
a:\xd4A\xe1\x05\xe2^><\xe7\xc4\xeeQJ\x90=\xe93E\x8f0\x8e\x1cDV\xe5\x06D\xe8\x1eb-g\x18vUpDjp
\xdd\xe7R\xcc\xd23<\x16\x9c\x8cy\x80L\xaaN\xc4\xd6\xcb\x82)\xfa\x848c)Fun\xcf\xb2\x87I\x03\x
da\x17\x12\xe5\x7f\xfc\xa6u\xbcn\xd5\xa8\t\xad\x1d\x16\xf2\x80\x82\x13\x81\x947\xe7\x1c\xdb
\x0bE\xe2~\x8dw,\x89\xa7S_%#\xbd\x81Z\x9c\x02\x80\x98\xb8i\xfa\x8b\x19\xdah\xea\x01cNE\x9a\x
93['
```

```
--------------------------------------------------------------------------------------------------
----------------------
mensagem inicial == mensagem final ?:  True
--------------------------------------------------------------------------------------------------
----------------------
```

Exercício 1.d.

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 1.b..

Criterion

O critério $C\kappa, n$ define quais vetores de chaves públicas **p** são considerados válidos. O **Receiver** deve criar um vetor **p** onde:

- Algumas entradas são chaves públicas "boas" (as que correspondem às mensagens que quer receber).
- O resto das entradas são chaves públicas "más" (geradas para manter a segurança do protocolo).

O critério $\mathcal{C}_\kappa$ depende da estrutura algébrica usada pelo mecanismo de geração de pares de chaves. Vamos considerar:

- Cada chave pública, válida ou não, é codificada por um inteiro em $\mathbb{F}_p^*$.
- Uma matriz de "rank" completo $\mathsf{A} \in \mathbb{F}_p^{*n\times(n-\kappa)}$ e um vector $\mathsf{u} \neq 0 \in \mathbb{Z}_q^{n-\kappa}$; estes elementos são gerados por um XOF a partir de uma "seed" ρ . A "seed" é aleatoriamente gerada e os restantes elementos são construídos com o XOF, por tentativas, até se verificarem as condições exigidas.
- O critério tem a forma de um sistema de equações lineares $\mathsf{p} \times \mathsf{A} = \mathsf{u}$.

In [16]:
```python
import numpy as np

class CknCriterion:
    def __init__(self, kappa, n, q):
        self.kappa = kappa
        self.n = n
```

```
        self.q = int(q)
        self.seed = np.random.randint(0, 2**32)
        self.A = self.generate_A()
        self.u = self.generate_u()
        self.Fp = GF(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 = [x[0] if isinstance(x, tuple) else x for x in p]
        print("p_values:",p_values)

        # Converter p_values para um vetor no corpo finito Z_q
        Zq = GF(self.q)
        p_vector = vector(Zq, p_values)

        # Calcular p * A no corpo finito Z_q
        A_matrix = matrix(Zq, self.A)
        pA = p_vector * A_matrix

        # Verificar se pA é igual a u
        u_vector = vector(Zq, self.u)  # Converter u para um vetor no corpo finito Z_q
        return pA == u_vector

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

Provider

In [17]:
```
class Provider:
    def __init__(self, pk, sk, n_mensagens):
        self.pk,self.sk = gen_keys(lambda_bits) #(p, q, g, pow(g, s, p)) , s
        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, q)
```

Inicialmente:

- Gera chaves
- Gera mensagens
- Expõe apenas o número de mensagens, matriz A e vetor u

In [18]:
```
n_mensagens = 100
provider = Provider(None,None,n_mensagens)
```

```
----------------------------------Geração de chaves:-------------------------------------
Parâmetro q gerado: 473425505304861110036818096476948165659
p não convergiu, novo valor de q:  651747836086344281351954319461271614363
p não convergiu, novo valor de q:  601471082612323349939029982096492979499
Parâmetro p gerado: 215094375779393975774016574623634751823852143403715845903541108840829929
454124348407857493305682026124217704269568959894133327956157057489001443360933372108916758307715654357662199315540180057042903829181821796856270220038252764543831080057210955536155331489709707943328114310722305101161382017909336692477010926194519409200945692023354820065462171674296319593126392304191602082328911110602149965552157982964433975343717031893968879617
gerador (g): 3
chave privada (s): 546348918973747653366047085971850553898
-------------------------------------------------------------------------------
```

Receiver

In [19]:
```
import hashlib
from random import sample

class Receiver:
    def __init__(self, k, n_mensagens, q):
        self.k = k  # Número de chaves privadas geradas
        self.n_mensagens = n_mensagens
        self.q = q
        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"""
        vetor_pk = [0] * self.n_mensagens
        vetor_sk = []
        for i in range(1,self.k+1):
            pk,sk = gen_keys(lambda_bits) # (p, q, g, pow(g, s, p)), s
            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, [self.p[i][0] if isinstance(self.p[i], tuple) else self.
```

```
        # 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 p
        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 a chave "má"
            p_mau, q_mau, g_mau, gs_mau = self.gen_mau_keys()
            self.p[idx] = (p_mau, q_mau, g_mau, gs_mau)
        return self.p

    def gen_mau_keys(self, max_attempts=100):
        """Gera uma chave 'má' no formato (p, q, g, g^s)"""
        attempts = 0
        while attempts < max_attempts:
            # Gerar q_mau como um número primo aleatório
            q_mau = random_prime(2^128, False, 2^127)

            # Tentar encontrar um k tal que p_mau = q_mau * 2^k + 1 seja primo
            for k in range(1, 10):  # Limitar k a um intervalo razoável
                p_mau = q_mau * (2^k) + 1
                if is_prime(p_mau):
                    # Criar o corpo finito F_p_mau
                    F_p_mau = GF(p_mau)

                    # Encontrar um gerador g_mau do grupo multiplicativo F_p_mau^*
                    g_mau = F_p_mau.multiplicative_generator()

                    # Gerar a chave privada s_mau aleatoriamente
                    s_mau = randint(1, q_mau - 1)

                    # Calcular g^s mod p
                    gs_mau = pow(g_mau, s_mau, p_mau)

                    return p_mau, q_mau, g_mau, gs_mau

            attempts += 1

        raise ValueError(f"Não foi possível gerar p_mau após {max_attempts} tentativas.")

    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("-----------------------------------------------------------------------")
```

Escolhe o número de mensagens que pretende receber do Provider tendo acesso ao número de mensagens disponíveis

In [20]:
```
k = 20
receiver = Receiver(k,n_mensagens,provider.pk[1]) # Receiver escolhe o conjunto I já na sua
```

```
----------------------------------Geração de chaves:-------------------------------------
Parâmetro q gerado: 468601276260083369395618784167366617347
p não convergiu, novo valor de q:  456386142030968830373266851338520761167
Parâmetro p gerado: 265099183591794405915695160300324671560308897386636795244324099854230498
456089622562903658082260194971043213337871005267171027971289906551786948653939748720067868600
800479922159552342898170573395675455235973477916251040700653028392327787125589008627853289194
860421892632520452472722179302596186154420596764473683713667891 3
gerador (g): 3
chave privada (s): 39822229966415039273058785286330073985
----------------------------------------------------------------------------------
----------------------------------Geração de chaves:-------------------------------------
Parâmetro q gerado: 658316136949427614649801826651412869359
Parâmetro p gerado: 463806833069585745556025376202561287641581594586099389844441258975005969
562883966080667802323056662883838028224129322941981529593761915064837726153257653759010598779
794399229307595744907705736445233618101383386694148302863306373290841879713145710365967614406
738385729139419465393504664946383514152437204822507901156587483281476520332976244408281538759
304595512387950219250784620944669624770891486864781115651471560561833242952602883948069046420
411511133782312382114467596742823104019288138697279998600286242000091897045383241271532284279
293356957237249
gerador (g): 3
chave privada (s): 34334714464975644886445816734786514232 3
----------------------------------------------------------------------------------
----------------------------------Geração de chaves:-------------------------------------
Parâmetro q gerado: 600562632040476496974591432613102259701
Parâmetro p gerado: 230606996058570763602532603701088688116214291571823225911710471859323325
134633549826175043471709843907297584355760839737277921650457613807151399941930593690239416869
471915569736668719807001226150802216167998974139368620458071003325729931710222544248868634794
557084751395792165086254113017794034700324300156854442735056634281627229828140864994780757940
291289033721911116329740938833097624331114752118263412315287749106954251636156465613809647617
gerador (g): 3
chave privada (s): 353368548385613739706645266031680521474
----------------------------------------------------------------------------------
----------------------------------Geração de chaves:-------------------------------------
Parâmetro q gerado: 615437303557669889369409730827750125603
p não convergiu, novo valor de q:  389821515904532788184485108681923467591
p não convergiu, novo valor de q:  361824098607611558860307177687820828083
p não convergiu, novo valor de q:  665289345282220409691668246927379559501
p não convergiu, novo valor de q:  557790163272807787109414619717124794047
Parâmetro p gerado: 606116446633337267089794535761229255337012049424707796537754769050583126
749566512404206221690374414904389381427031313204182239740944580543152944446510532532662517700
392370953118644946706275968851125765838755273062027757038779553700016562879187476671150137639
159660939924628822632747666385700114363156821619336176116696295671999983498467513825434941981
954544503955926653337 7
gerador (g): 3
chave privada (s): 25555781192704580403853278163118677851 3
----------------------------------------------------------------------------------
----------------------------------Geração de chaves:-------------------------------------
Parâmetro q gerado: 637962409690365289767631651711072663327
Parâmetro p gerado: 216635850667806208493404281713859596051616575765474328612441116048951513
636995766236716738906617247913619033114879515610092743988073614440907686244623996218219805578
973052056709845496016343164388635665441657010450047070811071040754724485935207522265064567388
458276773165353966371497145768457677450193201631406141212288131744399092675017315490952291217
8014795930145914 89
gerador (g): 3
chave privada (s): 19834113053857732560386088203989210004 3
----------------------------------------------------------------------------------
----------------------------------Geração de chaves:-------------------------------------
Parâmetro q gerado: 34738188596448192130065934181839297837 1
Parâmetro p gerado: 200702072448875520080300684743307751780364348774070432898765511411134093
485110886640897222325683456911180257864103926100308485660648719763668836222509008414278978968
068179610055421979634597371154360910646733120740844709091916509708290899305333762711171812089
934511031251957363765036102349222752460976625251673600903758410454760985136578626329701151374
796356782763394061623849148579168317541025120100676534273
gerador (g): 3
chave privada (s): 64469076890246090008274633860416453612
```

```
---------------------------------------------------------------------------
----------------------------------Geração de chaves:----------------------------------
Parâmetro q gerado: 55514884902457867025283402365969914 2029
Parâmetro p gerado: 368192316910910542008605213937899235372331776489507060732606162161899314
552876420897722167345667413422864351050837039619816126428365549001055194487386831824705575 57
318192874711016959424920717375941949741611729969582917531460406327866447689129518820247415 52
359441312905667259034047399198836005379402591342166305835129916166937095127364829939164456 42
6120085067423285249
gerador (g): 3
chave privada (s): 703176922192825484067150573253979869 23
---------------------------------------------------------------------------
----------------------------------Geração de chaves:----------------------------------
Parâmetro q gerado: 67943560935630705978872611721747586 7523
p não convergiu, novo valor de q:  573036012623438550261426996858117162361
p não convergiu, novo valor de q:  646403578903946875851245009924942437061
Parâmetro p gerado: 300379078603350560642792193697399723918839240997537400985844789498383416
365993805250609154164723757721306407194929325823167947261854863448549905864419778558739019 43
941678100569593086227041732886330054090253442621050645620589089845452568167456497718490911 12
749967222599174438768582628563042553575682111903257550483166855169
gerador (g): 3
chave privada (s): 493483774982927187919444285799807421619
---------------------------------------------------------------------------
----------------------------------Geração de chaves:----------------------------------
Parâmetro q gerado: 56839801250452137547850526044047799 4071
Parâmetro p gerado: 204361040989752573751662836745415271257662311110098629795729383733448123
673796120608663425059955370328811867919173463609857742938332135395376998314493903588632684 99
161663143632362015462664775118081975706428207533185261634105284321115348194809514491711031 60
930751189255989343000628391424678017586721899604353737703131658403581364181546612086768926 73
gerador (g): 3
chave privada (s): 743952287044945809963444001931313403 77
---------------------------------------------------------------------------
----------------------------------Geração de chaves:----------------------------------
Parâmetro q gerado: 66886061116453870679853032036798572224 7
p não convergiu, novo valor de q:  650468954555024072640925805945181960581
Parâmetro p gerado: 150976865075923373813126897413558941094709328567201746974646846893425173
690939962406943560914163070941111563466023444866795416356250905106996057612907645462580230 75
646943883921648378067254460821119373840009983887288396391034026041514772124623583685567979 14
966436753708325897679032647093085552912307724240848968147607432525089279325989348382596783 55
510538036817311999614302434031822987662097335233507338381807818233658382994727991609530305 4
575548676628613375894277324 9
gerador (g): 3
chave privada (s): 575314222048280470050622510417212979 26
---------------------------------------------------------------------------
----------------------------------Geração de chaves:----------------------------------
Parâmetro q gerado: 37108102974349871886863917604302256685 1
Parâmetro p gerado: 102231217396686438031657868999876699304802287840579281377155112045671434
113743825128804374987926434031429551384755260181819363411431123708958502896335756268120823 83
077134591666370927378784354805753499026587961066307754362728267179275100939002241451600793 83
978194603758034137607814151111490112537503407527660049797689491449662382939363081712405042 60
126809675141872474390263668989185418112327425107951617
gerador (g): 3
chave privada (s): 313840771016016143770368078138183248127
---------------------------------------------------------------------------
----------------------------------Geração de chaves:----------------------------------
Parâmetro q gerado: 57305896641689976139998891476613215926 1
Parâmetro p gerado: 785968147124446979860843381093551961738478310393472196713345984594980575
645861361377925670380782301426287196373487351563183849277638092385618430286712036958122603 80
202100223746373300859978266622912681274349607554415384236162592242070710375548057841778470 37
500924112468499277103222149247621800753068450638135702811327775626070019937316980654 09
gerador (g): 3
chave privada (s): 147233377015877404139868773789538801097
---------------------------------------------------------------------------
----------------------------------Geração de chaves:----------------------------------
Parâmetro q gerado: 58284803681796542867600043872973319 6493
Parâmetro p gerado: 336744124085055685128272699163897677297821403922367344247344238687266117
559981116692101921673682755037339604086447240980482041138411497005067760602903733601732013 88
```

532409536737947814679069256545533678911630328350106919234804104382409561374791880690974819685680735239092868707991616863733742850551875345458539874621366808892430033095024589492569925962001639312575165616310198568846811770522968561102595135897 7
gerador (g): 3
chave privada (s): 24369066587595607576662159599109113012 1

----------Geração de chaves:----------
Parâmetro q gerado: 42547081786004332748374253897804430802 1
Parâmetro p gerado: 54819344994734641657763569302355016645095656891490668137945641465915136686805319666658234850262779442279638830575368558064090336026504516437525040471951403812029949957215915954074673379888895656135002231474546166259507315624094160688089443570459852260825464546606444831793940675909617916440389063737157776155813791256016927065314552029577997957876517664886057726262302530544531099680055452439476119623588765076100253669853199492534831167610180620779531 3
gerador (g): 3
chave privada (s): 28873741311053320790081142732738716663 2

----------Geração de chaves:----------
Parâmetro q gerado: 57673721481709484813912870013206113447 1
Parâmetro p gerado: 93791278530561233867740258106995977555058765937657688076688212721877659202327149631657431448500722982482438147110764736667984805894239919526834194710638597189341093658701149169100284044430312973857938134621798809704448211844134354789220286405409573175917767660224620121541213604098408244822851012975378187421732734813005339327468783027867077629768554764930032843751679981790794551872307124678300464932625795735826445893633 3
gerador (g): 3
chave privada (s): 65842499376991967703800559498306415128 1

----------Geração de chaves:----------
Parâmetro q gerado: 639232617026483874067874552423254280287 1
p não convergiu, novo valor de q: 501031297950676357270095335975068326511 1
Parâmetro p gerado: 151112472538208090621942149825335140067887422969104597608093340446764492011076256178657169712841643614765124456860861310700667674381092516845437117968110666286964426899322221767545240001766895710239862211836225959040648574190782368894940025155085381380145784353504150975115617582282275219993721021302464933294035559990252541310096393125405369044628361217 7
gerador (g): 3
chave privada (s): 17735521666414614521055034030121319328 2

----------Geração de chaves:----------
Parâmetro q gerado: 65432014409712875387686809212016210963 7
Parâmetro p gerado: 88308430292390889382733944533685081026000092876835205659606567849690303389685158801837377414703575030634340340996090503727971607640013481351141192969826305267218431792185408192097390130014069298895155586864941529797841722801480509791086751610846408841751191425596067750784839471194839704842326302975056054821112136114602106242678270104738091363522158086602939176399543193628602433931485376229832004503947888979215730426902993045587789890669082278008553504006891406050603842721365434008218394555257580525027461779981175934019716370097412679508791388828686225211106320399956267431469098186108358236097740 9
gerador (g): 3
chave privada (s): 62018927454818381985298315087734812360 4

----------Geração de chaves:----------
Parâmetro q gerado: 49710217161538635059819039416218722110 3
Parâmetro p gerado: 59970974718875660043963356942398337003414771154099216670877548291777008272642822344619045125778609542307225133587189626421309314449871029277308710034922647390897404733184576494206384814217006158211121870860880163909234991776155458580131176355368849297243010487683797880016972214602417327663636124437015739525566407974153477892096773824915925983631738142 3
gerador (g): 3
chave privada (s): 33802175906838244659207326741216129153 9

----------Geração de chaves:----------
Parâmetro q gerado: 47749746152456332615943816000917796477 9
Parâmetro p gerado: 15447360784354040545141760395877417168184500446827311534401211028724940297710224668750067905519022954500790687323681842802320389924516127412441169768527972127880822453331422704239238840459921045779652220536989877824472569418604836348765782576286392800194448984356733346441590622431390003631784971992840870921920440278299775818443651333007802705338 3

```
994582857245014874469499269762790579884004355387467506056016034550465690069225455445887332614719454685079325475972760272822189767265197012071661455230780867346433
gerador (g): 3
chave privada (s): 250285680461530541799587819427748753442
--------------------------------------------------------------------------------
----------------------------------Geração de chaves:----------------------------------
Parâmetro q gerado: 364094411918235323009675669672989103539
Parâmetro p gerado: 346505309955002100551797139585257814314022917225551650350768110484710107868426002913313001179050093978959656722251194663558715143147004091438695356566985478487659706274843534869717326903257303451055400433548486348929504073657320114889106796340406031555223916461781844746002175872289834348002289288539977257592940209806875033 7
gerador (g): 3
chave privada (s): 32640409292152014032054013345466368468 4
--------------------------------------------------------------------------------
```

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

In [21]:
```
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 $e\colon \{1, \kappa\} \to \{1, n\}$ 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 [22]:
```
receiver.print_info()
```

```
----------------------------------------------------------------------
Seleção I: [7, 74, 80, 57, 2, 29, 3, 20, 4, 67, 15, 28, 10, 98, 40, 35, 61, 8, 55, 64]
----------------------------------------------------------------------
Função de enumeração e: {1: 2, 2: 3, 3: 4, 4: 7, 5: 8, 6: 10, 7: 15, 8: 20, 9: 28, 10: 29, 1
1: 35, 12: 40, 13: 55, 14: 57, 15: 61, 16: 64, 17: 67, 18: 74, 19: 80, 20: 98}
----------------------------------------------------------------------
Segredo s: 1626447042
----------------------------------------------------------------------
Chaves privadas s_i: [398222299664150392730587852863300739 85, 343347144649756448864458167347
865142323, 353368548385613739706645266031680521474, 255557811927045804038532781631186778513,
198341130538577325603860882039892100043, 644690768902460900082746338604164536 12, 70317692219
282548406715057325397986923, 493483774982927187919444285799807421619, 743952287044945809963 4
4400193131340377, 575314222048280470050622510417212979 26, 313840771016016143770368078138183 2
48127, 147233377015877404139868773789538801097, 243690665875956075766621595991091130121, 288
737413110533207900811427327387166632, 658424993769919677038005594983064151 28, 17735521666414
614521055034030121319328 2, 620189274548183819852983150877348123604, 338021759068382446592073
267412161291539, 250285680461530541799587819427748753442, 326404092921520140320540133454663 6
84684]
----------------------------------------------------------------------
Vetor p (com chaves públicas mapeadas): [0, 0, (265099183591794405915695160300324671560308 89
738663679524432409985423049845608962256290365808226019497104321333787100526717102797128990 65
517869486539397487200678686080047992215955234289817057339567545523597347791625104070065302 83
923277871255890086278532891948604218926325204524727221793025961861544205967644736837136678 91
3, 456386142030968830373266851338520761167, 3, 103008136187908351401720311287907291050006374
079978479625359689173550807975371698533359150948566332021851875953051212709069913707057166 67
841114505064251682587751780369619696508131118398225848033070603285641723654790269070629735 20
183588633618313883414997952641886838218230722200500448182536260358993435243488343409842519 0
4), (463806833069585745556025376202561287641581594586099389844441258975005969562883966080 667
802323056662883838028224129322941981529593761915064837726153257653759010598777943992293075 95
744907705736445233618101383386694148302863306373290841879713145710365967614406738385729139 41
946539350466494638351415243720482250790115658748328147652033297624440828153875930459551238 79
502192507846209446696247708914868647811156514715605618332429526028839480690464204115113378 23
123821144675967428231040192881386972799986002862420000918970453832412715322842792933569572 37
249, 658316136949427614649801826651412869359, 3, 326525563543305843850213694562181336929672 2
978886282257182640538696917523426641115296147836213192062682671925884721086867217038402624 97
816301333405906970626071314709943753053997990766039608862811029165500883945685060937804338 33
683354767161394119603828339321141791744475529518936299541445758523330012134456470811940018 83
662315608479514641334504308939986259448671245155920942182895638151463824103663998976933728 67
924693778095658597689878695599029942933442906222281996340478873903379989055248084126606942 62
720134584910883605022280692768182398065539512 41), (230606996058570763602532603701088688116 21
429157182322591171047185932332513463354982617504347170984390729758435576083973727792165045 76
138071513999419305936902394168694719155697366687198070012261508022161679989741393686204580 71
003325729931710222544248868634794557084751395792165086254113017794034700324300156854442735 05
663428162722982814086499478075794029128903372191111632974093883309762433111475211826341231 52
877491069542516361564656138096476 17, 600562632040476496974591432613102259701, 3, 99737098153
617517067425667550212904969240736122111607138607372042302494803186351432769883281975949350 52
438974393376694473654125333377848456211493058536065636725203659989865507395593241377205058 38
348892622509849130923692739631545876480339305031018097140167752774721983536700640068834973 77
650783974902553292272839148309758411587413680819466098537320574559184258042104767132089252 96
788097991588637499310859889737107710461681266972980972353020083 4), 0, 0, (606116446633337267
089794535761229255337012049424707796537754769050583126749566512404206221690374414904389381 42
703131320418223974094458054315294444651053253266251770039237095311864494670627596885112576 58
387552730620277570387795537000165628791874766711501376391596609399246288226327476663857001 14
363156821619336176116696295671999983498467513825434941981954450395592665333 77, 5577901632728
077871094146197171247940 47, 3, 284772284084226181211493855313472420354954594417872921349074 1
849320712869422603540916259651375090377857145532647858685052224953550172906828256719195166 99
382769970348046746872387858003063993248821882250827922967453668283197911634831393919166501 44
704984656851875134010052205261691121398133283053340400542753690392103077605257982759602466 85
664871809725384571353398766803138 5), (216635850667806208493404281713859596051616575765474328
612441116048951513636995766236716738906617247913619033114879515610092743988073614440907686 24
464239962182198055789730520567098454960163431643886356654416570104500470708110710407547244 85
935207522265064567388458276773165353966371497145768457677450193201631406141212288131744399 09
267501731549095229121780147959301459 1489, 637962409690365289767631651711072663327, 3, 485657
491874900032234270712467221881050187980851801043219715182476766159892046204868192443644220 45
043298678113249302256757586411488705445405550782347683874209138806657077373211176434322853 88
711398495721163301798906155368763154045989875789652851476569318889910785906754808659255269 42
```

67782383778538191743409491571267081114991098669171922864906492020283251884285758682588 0,
(200702072448875520080300684743307751780364348774070432898765511411134093485110886640897222 3
25683456911180257864103926100308485660648719763668836222509008414278978968068179610055421979
63459737115436091064673312074084470909191650970829089930533376271117181208993451103125195736
37650361023492227524609766252516736009037584104547609851365786263297011513747963567827633940
61623849148579168317541025120100676534273, 34738188596448192130065934181839297837 1, 3, 12544
31813057378867311837204454100974119180954868308790237921748248741419957595441506932102868427
03219825980473822522893852693617972058344360367168910554721684991743074777105890663117226061
64939094647410807828010054636297665230079116472452436045295890019246118205201709447985644890
97189854574974200758518660574980711632554420321773954318552565897468952341325700803755524237
98038090617199694062074666478948259), 0, 0, 0, 0, (36819231691091054200860521393789923537233
17764895070607326061621618993145528764208977221673456674134228643510508370396198161264283655
49001055194487386831824705575573181928747110169594249207173759419497416117299695829175314604
06327866447689129518820247415523594413129056672590340473991988360053794025913421663058351299
16166937095127364829939164456426120085067423285249, 55514884902457867025283402365969914202 9,
3, 28667935640297504942691700138624101964164359300554778993397178161273277070722429698623096
33521156843603456037028438452517078490226694469949915278762999866098819609031981790207497609
74159743146967403807670347939380695047899953469546522350578637507415343340098472940775262737
98995629038090544611035533783752675700855558315717798227910863341651925148984825382444380178
77), 0, 0, 0, 0, (30037907860335056064279219369739972391883924099753740098584478949838341636
59938052506091541647237577213064071949293258231679472618548634485499058644197785587390194394
16781005695930862270417328863300540902534426210506456205890898454525681674564977184909111274
99672225991744387685826285630425535756821119032575504831668551 69, 64640357890394687585124500
9924942437061, 3, 29269711455830631685105966360884211516788491024117707912129477162624284930
51622257145002539020582066549143976659404307266552217796411135752029260051418899200520263085
81075834199703814743507155912410120735537163258430448130678323736445196758136535758155880827
26913025260004907783016561178503245223762924591597927522139434 13), 0, 0, 0, 0, 0, 0, 0, (204
36104098975257375166283674541527125766231111009862979572938373344812367379612060866342505995
53703288118679191734636098577429383321353953769983144939035886326849916166314363236201546266
47751180819757064282075331852616341052843211153481948095144917110316093075118925598934300062
83914246780175867218996043537377031316584035813641815466120867689 2673, 56839801250452137547 8
505260440477994071, 3, 72913851249874518567021970792858666851822701662217261268697570452989 3
43066314832958305123767085703837985700343895732112721054636835215462384331309631185487414846
19101163584551625268977623731401204785756304916040430475126533539471263844624778370414430540
18278988636269447560359022486921277880576885963542164668085102921920198661795965275383414264
68), (15097686507592337381312689741355894109470932856720174697464684689342517369093996240694
35609141630709411115634660234448667954163562509051069960576129076454625802307564694388392164
83780672544608211193738400099838872883963910340260415147721246235836855679791496643675370832
58976790326470930855529123077242408489681476074325250892793259893483825967835551053803681731
19996143024340318229876620973352335073383818078182336583829947279991609530305457554867662861
33758942773249, 65046895455502407264092580594518196058 1, 3, 14050017287940350604573359354567
20325915355377811555676811970072461690627424508326946376583240015757363314226452127759726332
34002552416975928166176306441490336312806284891618563276466088228230293276772098762495676351
77212279690251930539838411361080844109701379462557438028653272565067848197841827875508663371
50054564402490081594981126690464968648849540429049263731213197748651832135518108677413240929
32319072757402620302755978980123756194702021437867635202919859547270), 0, 0, 0, 0, 0, (10223
12173966864380316578689998766993048022878405792813771551120456714341137438251288043749879264
34031429551384755260181819363411431123708958502896335756268120823830771345916663709273787843
54805753499026587961066307754362728267179275100939002241451600793839781946037580341376078141
51111490112537503407527660049797689491449662382939363081712405042601268096751418724743902636
68989185418112327425107951617, 37108102974349871886863917604302256685 1, 3, 81619720786566550
64028326267512106512166663200310355670607996700201220418818070975832139925188871818276712732
75658863199922218613016923847385395401194616090506968300062564808292854604897496806018546478
14592212497895872724526508203141299539235825258080175482108915493856933638504287250521110337
61108723535232534271334282167524311667312739576308057215403703921373897450632287096916005274
0959093586472339), 0, 0, 0, 0, (78596814712444697986084338109355196173847831039347219671334 5
98459498057564586136137792567038078230142628719637348735156318384927763809238561843028671203
69581226038020210022374637330085997826662291268127434960755441538423616259224207071037554805
78417784703750092411246849927710322214924762180075306845063813570281132777562607001993731698
065409, 57305896641689976139998891476613215926 1, 3, 38719499936104878146948567582707971139 57
85667913369585274105728750511861860907241676241464013930590593466098588065327167422943873516
26558424885663190824996820907549096702060239161446884114104425983660090137274068828720390587
55031990688732924299654072482535829571505169305534758525330531790435394210306246635800545771
58865114758612898452985888), 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, (3367441240850556851 2
82726991638976772978214039223673442473442386872661175599811166921019216736827550373396040864
47240980482041138411497005067760602903733601732013885324095367379478146790692565455336789116

30328350106919234804104382409561374791880690974819685680735239092868707991616863733742850551
87534545853987462136680889243003309502458949256992596200163931257516561631019856884681177052
29685611025951358977, 582848036817965428676000438729733196493, 3, 172916325166441953954457534
44247465820975481600121269419337310498985257717523354756308443176382992837881634212305197478
96373903036996388438622231864575146022274992809018903361006868129552523988476352594788059377
45495323354627666595536393439789385989881178811653256259654216829210688515203961013855936388
47517597912022818308615942342999160073306897992795772847245946550091754811504129461632043487
38444931190770), 0, (54819344994734641657763569302355016645095656891490668137945641465915136
68680531966665823485026277944227963883057536855806409033602650451643752504047195140381202994
99572159159540746733798888956561350022314745461662595073156240941606880894435704598522608254
64546606444831793940675909617916440389063737157776155813791256016927065314552029577997957876
51766488605772626230253054453109968005545243947611962358876507610025366985319949253483116761
018062077953, 42547081786004332748374253897804430802l, 3, 87702603192908039785851554520745567
72407276510523769333900176629335256545399934228252563557448501880146639162197996202893057950
94320932108825707281710492511261531945285587249549725339949476187675628112506732660265757128
27284196729090712010790266450812046425412537221454133115145052357425656799911822256925103183
42055911331037911910662114453858118989739318973365140248965637396326729934462105648856876682
05535296256029599775886779715822462058317083369O), 0, 0, 0, (937912785305612338677402581069 9
59775550587659376576880766882127218776592023271496316574314485007229824824381471107647366679
84805894239919526834194710638597189341093658701149169100284044430312973857938134621798809704
44821184413435478922028640540957317591776766022462012154121360409840824482285101297537818742
17327348130053393274687830278670776297685547649300328437516799817907945518723071246783004649
32625795735826445893633, 57673721481709484813912870013206113447l, 3, 47211955200687564581861
45274610476633502507459770520190820407239125582040391279386063386037827302917569602274291581
26693129669696303690959505471229883006475909118079197433380100801817723688987618005727718998
93283015078540849817331808786287242707519160258847810044564100426465655028847097417917905179
01650143911444074185197599636183126312548077659314157704196487994180297310702946439800340644
61109331055770129923698270669801, 0, 0, (151112472538208090621942149825335140067887422969104
59760809334044676449201107625617865716971284164361476512445686086131070066767438109251684543
71179681106662869644268993222217675452400017668957102398622118362259590406485741907823688949
40025155085381380145784353504150975115617582282752199937210213024649332940355599902525413l0
09639312540536904446283612l7, 501031297950676357270095335975068326511, 3, 155866383120528871
69491360259952771112842937468222137930643508621131183763562955620774508758529744828533518466
73238972400843910442227107533133287983119608854057865400689173462407666181699397974974566225
40739372691411067684200949675591662192599579071945188055210633844112195002257961058489570328
67004330248611040632041551380505677463858791218640388050767), 0, 0, (883084302923908893827333
94453368508102600009287683520565960656784969030338968515880183737741470357503063434034099609
05037279716076400134813511411929698263052672184317921854081920973901300140692988951555868649
41529797841722801480509791086751610846408841757191425596067750784839471194839704842326302975
05605482111213611460210624267827010473809136352215808660293917639954319362860243393148537622
98320045039478889792157304269029930455877898906690822780085535040068914060506038427213654340
08218394555257580525027461779981175934019716370097412679508791388828686225211106320399956267
43146909818610835823609774O9, 654320144097128753876868092120162109637, 3, 623118353751748022
99635326214823196715936547297448292121097071127517233906394697956056453019691937181520951771
87347094002124803925777941655182387331314865087779334664260812524711172519934810177140201916
37833088655555784904029854701357189022095968138097883248082256448910815110085755771264670866
64010422254650841814941846510544387633940832366472426630786217377049442168121993754188060822
13412177606930581942740088791241686231601820369031899565749891723805811675742082021724173793
66642471390776094416739004729884894735650958034376984335361414462080306561087451874164607896
84726107580520528927476589058761З), 0, 0, 0, 0, 0, 0, (599709747188756600439633569423983370O
34147711540992166708775482917770082726428223446190451257786095423072251335871896264213093144
49871029277308710034922647390897404733184576494206384814217006158211121870860880163909234991
77615545858013117635536884929724301048768379788001697221460241732766363612443701573952556640
79741534778920967738249159259836317381427З, 49710217161538635059819039416218722110З, 3, 5508
16998750700052171960152802632248711019995599083820598745790403060846215260583251691926245258
81730465060004646471270615142187697134622506398870850011758700119004410111324597601265491255
15273423929471332717197885666259624646784197229241804004923614375003048096760960084238368120
42216648145630484614905464249543167517453750794270095529970350956238495770З), 0, 0, 0, 0, 0,
(15447360784354040545141760395877417168184500446827311534401211028724940297710224668750067900
55190229545007906873236818428023203899245161274124411697685279721278808224533314227042392388
40459921045779652220536989877824472569418604836348765782576286392800194448984356733346441590
62243139000363178497199284087092192044027829977581844365133300780270533839945828572450148744
69499269762790579884004355387467506056016034550465690069225455445887332614719454685079325475
97276027282218976726519701207166145523078086734643З,
477497461524563326159438160009177964779
, 3, 11946642993078642531164997847228264706733611241163929481406980114319921650780415057022 4
60626301687234223489856312858248484514578646695407910606710823827019395343076543615261005937

```
8621440909097183044676860592086361614506489846754860553688211445613232432895950774594681281325494884595444494856723438896341499118232305962501245200223225210452517853168250702258796042996484099132319414105986771870365180671324930918813355025486922710949939620229015877745105479992326616107794117523637073867332673227316634356518767), 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, (34650530995500210055179713958525781431402291722555165035076811048471010786842600291331300117905009397895965672225119466355871514314700409143869535656698547848765970627484353486971732690325730345105540043354848634892950407365732011488910679634040603155522391646178184474600217587228983434800228928853997725759294020980687503378640944119182353230096756696729891035398, 3, 7332320514824401795356633306658510535595739427845862494631373495028502012845536579869676651758448643307758672919949605119971859275221513409018657367397877256032001809080325761980155858075500229550137497706621623912852266045042677596612234826352409051890861665686858727516782304949058872689960044280759859089781452580476407700), 0]
----------------------------------------------------------------------
Tag de autenticação τ: b'4\xd25Eq\x81\xb0\r\xb1\xdd\x99X\x07\x84Qa\xfa\xa8\xad\x1c\xeb\x94\xa4\xdc\x82S-\x14\xc1\xea0`'
----------------------------------------------------------------------
```

Em seguida completa a definição de $\mathbf{p}$ atribuindo às compompentes $\{\mathbf{p}_j\}_{j\notin I}$valores tais que o vetor de chaves públicas $\mathbf{p}$ 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.

Mais concretamente essa forma de completar o vetor é descrita na classe Receiver:

```
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, [self.p[i][0] if isinstance(self.p[i], tuple) else self.p[i] for i in filled_indices]) # 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 p[i] == 0) 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 "má" como uma tupla de 4 dimensões # Aqui, você pode usar a função gen_keys para gerar uma nova chave (p, q, g, g^s) # Ou, se preferir, usar valores padrão ou aleatórios para as chaves "más" p_mau, q_mau, g_mau, gs_mau = self.gen_mau_keys() # Exemplo de função para gerar chaves "más" self.p[idx] = (int(p_empty[i]), q_mau, g_mau, gs_mau) # Tupla de 4 dimensões return self.p
```

O código realiza a seguinte sequência de operações para completar o vetor p de forma que satisfaça a equação p×A=up×A=u no corpo finito Z_q:

- Definição do corpo finito Z_q usando `GF(self.q)`.
- Identificação dos índices preenchidos no vetor p usando `filled_indices`, que são obtidos a partir dos valores em `self.e`. Os valores correspondentes a esses índices são armazenados em `filled_values`.
- Criação das matriz `A_filled` partir das linhas da matriz A correspondentes aos índices preenchidos.
- Criação das matriz `A_empty` a partir das linhas da matriz A correspondentes aos índices vazios.
- Cálculo do vetor u' subtraindo a contribuição dos elementos preenchidos de u, ou seja, `u' = u - (filled_values × A_filled)`.
- Resolução do sistema linear $p_{empty} = A_{empty}^T . u'$ para preencher os valores desconhecidos de p (chaves "más").
- Preenchimento de p, colocando os valores de p_{empty} índices correspondentes no vetor p.

Finalmente o Receiver envia ao Provider a "tag" τ e o vetor $\mathbf{p}$

In [23]:
```
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 $\mathbf{p}$ não for aceite pelo critério então aborta o protocolo.

Se $\mathbf{p}$ 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 [24]:
```
def h_OT(r,y,t):
    """Hash pseudoaleatório h(r, y, t) com tamanho lambda_bits"""
    h = hashlib.sha512()
    r_bytes = r.to_bytes((r.bit_length() + 7) // 8, 'big')
    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()[:lambda_bits // 8]  # Truncar para lambda bits
    #print("h(r, y) hash:", full_hash)
    return full_hash
```

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

In [25]:
```
def f_p_OT(pk, r, rlinha):
    p, q, g, g_s = pk
    # Verificar se p é primo
    if not is_prime(p):
        raise ValueError(f"p = {p} não é um número primo. Não é possível criar o corpo fini


    # Calcular gamma e kappa
    gamma = pow(g, rlinha, p)  # γ = g^ω mod p
    kappa = pow(g_s, rlinha, p)  # κ = (g^s)^ω mod p

    # Multiplicar r_fp por kappa no corpo finito F_p
    result = (r * kappa) % p

    return (gamma, result)
```

In [26]:
```
def enc_fujisaki_OT(pk,x,t):
    # r gerado aleatoriamente com lambda bits
    r = ZZ.random_element(2^(lambda_bits - 1), 2^lambda_bits)

    # y = x XOR g(r)
    y = bytes(a ^^ b for a, b in zip(x, g(r)))

    # r' = h(r,y,tau)
    rlinha = h_OT(r,y,t)
```

```
    print(rlinha)

    # c gerado a partir do núcleo determinístico
    c = f_p_OT(pk,r,int.from_bytes(rlinha,"big"))

    #Devolver criptograma
    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 [27]:
```
import sys
if not provider.criterion.verify(completed_p):
    print("--------------------------------------------------------------------------")
    print("O vetor p foi aceite")
    print("p final:[ ")
    for key in completed_p:
        print(key)
    print("]")
    print("--------------------------------------------------------------------------")
    ciphertext_vector = []
    i = 0
    for public_key in completed_p:
        if isinstance(public_key, tuple):
            byte_length = (sys.getsizeof(provider.messages[i]))
            ciphertext_vector.append(enc_fujisaki_OT(public_key, bytes(provider.messages[i]
            i += 1
        else:
            raise ValueError(f"encontrada chave pública não primo: {public_key}")
else:
    print("O vetor p não foi aceite. Abortado")
```

p_values: [16346905232881361996093839879029244433729, 24665014097516468360373697625513320522
97, 26509918359179440591569516030032467156030889738663679524432409985423049845608962256290360
58082260194971043213337871005267171027971289906551786948653939748720067868608004799221595523
42898170573395675455235973477916251040700653028392327787125589008627853289194860421892632520
45247272217930259618615442059676447368371366789l3, 46380683306958574555602537620256128764158
15945860993898444412589750059695628839660806678023230566628838380282241293229419815295937619
15064837726153257653759010598777943992293075957449077057364452336181013833866941483028633063
73290841879713145710365967614406738385729139419465393504664946383514152437204822507901156587
48328147652033297624440828153875930459551238795021925078462094466962477089148686478111565147
15605618332429526028839480690464204115113378231238211446759674282310401928813869727999860028
62420000918970453832412715322842792933569572372l9, 23060699605857076360253260370108868811621
42915718232259117104718593233251346335498261750434717098439072975843557608397372779216504576
13807151399941930593690239416869471915569736668719807001226150802216167998974139368620458071
00332572993171022254424886863479455708475139579216508625411301779403470032430015685444273505
66342816272298281408649947807579402912890337219111163297409388330976243311147521182634123152
87749106954251636156465613809647617, 49266203683031139270997339861625750947073, 358336171503
69253930109795004212958926l7, 60611644663333726708979453576122925533701204942470779653775476
90505831267495665124042062216903744149043893814270313132041822397409445805431529444465105325
32662517700392370953118644946706275968851125765838755273062027757038779553700016562879187476
67115013763915966093992462882263274766638570011436315682161933617611669629567199998349846751
38254349419819544503955926653337l, 21663585066780620849340428171385959605161657576547432861l
44111604895151363699576623671673890661724791361903311487951561009274398807361444090768624464
23996218219805578973052056709845496016343164388635665441657010450047070811071040754724485935
20752226506456738845827677316535396637149714576845767745019320163140614121228813174439909267
50173154909522912178014795930145914891, 63928518349942420812052395587579926646091, 20070207244
88755200803006847433077517803643487740704328987655114111340934851108866408972222356834569111
80257864103926100308485660648719763668836222509008414278978968068179610055421979634597371154
36091064673312074084470909191650970829089930533376271117181208993451103125195736376503610234
92227524609766252516736009037584104547609851365786263297011513747963567827633940616238491485
79168317541025120100676534273, 37209929270973273912374354115657950987691, 1127637558180894838
30559241069091655470731, 62940334573236662353911792661512750503 9, 22319077986083940708819238 4
7613480358729, 36819231691091054200860521393789923537233177648950706073260616216189931455287
64208977221673456674134228643510508370396198161264283655490010551944873868318247055755731819
28747110169594249207173759419497416117299695829175314604063278664476891295188202474155235944
13129056672590340473991988360053794025913421663058351299161669370951273648299391644564261200
85067423285249, 17735002582936500122786361708114932222 33, 15131339678159231327086725966750 32
42707457, 35608107424230574185783651475137698952833, 11928352485987800699151739269703726868 5
313, 30037907860335056064279219369739972391883924099753740098584478949838341636599380525060 9
15416472375772130640719492932582316794726185486344854990586441977855873901943941678100569593
08622704173288633005409025344262105064562058908984545256816745649771849091112749967222599174
43876858262856304255357568211190325755048316685516 9, 14633091952150294254807992706165013149 5
3, 16484808226066359511581408425260008333377, 58601875683895985857976292096443441582849, 132
46645471777758005295557684233744038553 7, 14243262691972323441323054230855001884 3, 161839074
48405302840996563801425109660825 7, 29330229026698669153894243321278426782849, 20436104098975
25737516628367454152712576623111100986297957293837334481236737961206086634250599553703288118
67919173463609857742938332135395376998314493903588632684991616631436323620154626647751180819
75706428207533185261634105284321115348194809514491711031609307511892559893430006283914246780
17586721899604353737703131658403581364181546612086768926 73, 15097686507592337381312689741355
89410947093285672017469746468468934251736909399624069435609141630709411115634660234448667954
16356250905106996057612907645462580230756469438839216483780672544608211193738400099838872883
96391034026041514772124623583685567979149664367537083258976790326470930855529123077242408489
68147607432525089279325989348382596783555105380368173119996143024340318229876620973352335073
38381807818233658382994727999160953030545755486766286133758942773249, 41395777262051386679 1
6743809245792001409, 10141881860582749255549115318051889485 9, 51182709607265276360627342044
623288176897, 24893503058492543652652158155423806127 3, 44220000209668912948873742021607085 5
93857, 10223121739668643803165786899987669930480228784057928137715511204567143411374382512 8
04374987926434031429551384755260181819363411431123708958502896335756268120823830771345916663
70927378784354805753499026587961066307754362728267179275100939002241451600793839781946037580
34137607814151114901125375034075276600497976894914496623829393630817124050426012680967514 8
72474390263668989185418112327425107951617, 21738011017898402987759893466476404409409, 602553
37335840879644028181490552108724 7, 41393182027837544923794914534179266142 7, 759539755527332
17036625323622083632243 3, 78596814712444697986084338109355196173847831039347219671334598459
49805756458613613779256703807823014262871963734873515631838492776380923856184302867120369581
22603802021002237463733008599782666229126812743496075544153842361625922420707103755480578417
78470375009241124684992771032221492476218007530684506381357028113277756260700199373169806540
9, 47079780582140988615571415121328613955329, 15232805148782581190452917419163454950593, 101

032114242656031352712360480462947502 9, 718557800662050400950345727953441567669 77, 1086646735
336021885707162003781366985774 09, 210049748546323236998097900374996228993 93, 163258297304455
963660138125337043676184 97, 127357728482094016876097380716011115925 13, 435187908830282441242
700469893019911637 77, 255926369415038628640160650279914934583 3, 696105402200016572381715132 3
90670781069, 162958517405539346668839276867677408430593, 628113842692561806172467699138120 52
3127, 331055545981640733017530001074443383492 9, 336744124085055685128272699163897677297821 40
392236734424734423868726611755998111669210192167368275503733960408644724098048204113841149 70
050677606029037336017320138853240953673794781467906925654553367891163032835010691923480410 43
824095613747918806909748196856807352390928687079916168637337428505518753454585398746213668 08
892430033095024589492569925962001639312575165616310198568846811770522968561102595135897 7, 12
342823578745255123760908698850231383 89, 548193449947346416577635693023550166450956568914906 6
813794564146591513668680531966665823485026277944227963883057536855806409033602650451643752 50
404719514038120299499572159159540746733798888956561350022314745461662595073156240941606880 89
443570459852260825464546606444831793940675909617916440389063737157776155813791256016927065 31
455202957799795787651766488605772626230253054453109968005545243947611962358876507610025366 98
531994925348311676101806207795 3, 132403198071660013770509985511247947483649, 606841896451040
236787906557188400085795 3, 301487455424784166609504423213236737396 9, 937912785305612338677 40
258106995977555058765937657688076688212721877659202327149631657431448500722982482438147110 76
473666798480589423991952683419471063859718934109365870114916910028404443031297385793813462 17
988097044482118441343547892202864054095731759177676602246201215412136040984082448228510129 75
378187421732734813005339327468783027867077629768554764930032843751679981790794551872307124 67
830046493262579573582644589363 3, 525119461628012039684431443720189944852 9, 21416994806772124
708652763022654414432193, 151112472538208090621942149825335140067887422969104597608093340446
764492011076256178657169712841643614765124456860861310700667674381092516845437117968110666 28
696442689932222176754524000176689571023986221183622595904064857419078236889494002515508538 13
801457843535041509751156175822822752199937210213024649332940355599902525413100963931254053 69
0444628361217, 297555337135100075663919442498774579211 3, 393688465240593472954344856603468 63
03409, 883084302923908893827339445336850810260000928768352056596065678496903033896851588018 3
737741470357503063434034099609050372797160764001348135114119296982630526721843179218540819 20
973901300140692988951555868649415297978417228014805097910867516108464088417571914255960677 50
784839471194839704842326302975056054821112136114602106242678270104738091363522158086602939 17
639954319362860243393148537622983200450394788897921573042690299304558778989066908227800855 35
040068914060506038427213654340082183945552575805250274617799811759340197163700974126795087 91
388828686225211106320399956267431469098186108358236097740 9, 294920712532029695451662411340 44
323226753, 742663701946986979070299173831298179869, 133244742137658178827871494899166793525 1
3, 221143763791523881250600250041936266789 7, 915953143715963659764448826270484378373 13, 7954
455093521766786362021570193394995020 9, 599709747188756600439633569423983370034147711540992 16
670877548291777008272642822344619045125778609542307225133587189626421309314449871029277308 71
003492264739089740473318457649420638481421700615821112187086088016390923499177615545858013 11
763553688492972430104876837978800169722146024173276636361244370157395255664079741534778920 96
773824915925983631738142 73, 603916505800294501393972796288297944823, 137737573270701854691 94
05800282513556329, 435424244159862210214296240630629744747 3, 343145214545549962355860614546 3
972485969, 858503215234627756550699694060861477690 89, 154473607843540405451417603958774171 68
184500446827311534401211028724940297710224668750067905519022954500790687323681842802320389 92
451612741244116976852797212788082245333142270423923884045992104577965222053698987782447256 94
186048363487657825762863928001944489843567333464415906224313900036317849719928408709219204 40
278299775818443651333007802705338399458285724501487446949926976279057988400435538746750605 60
160345504656900692254554458873326147194546850793254759727602728221897672651970120716614552 30
780867346433, 456500455369510130822297515599323881967, 723529372028303718363692348280151571
2257, 233507282857636179559435824953112968159 3, 750986396004331898209661742743132976939 53, 1
714786969784071446879970343937428437561 7, 280983625399083548101052854904519253978 89, 2865387
951579789006191983775535705608753, 738708371235601790823813453827847479429, 5312404584668615
886504261184672728976153 7, 212317887570463271448259068555579917176 97, 482312304921478977134 7
724592468067105713, 375965815482248376100986275819427615143 69, 169717754967117560378129483 88
6314533990913, 376306106994482123681543730475285862228 9, 147523776748424421923492787052625 98
81337, 142331394085083405079071973866923545688 9, 353242423416111025847059535057353467985 7, 3
465053099550021005517971395852578143140229172255516503507681104847101078684260029133130011 79
050093978959656722251194663558715143147004091438695356566985478487659706274843534869717326 90
325730345105540043354848634892950407365732011488910679634040603155522391646178184474600217 58
722898343480022892885399772575929402098068750337, 264257127324870252024512533048919392405 7]

O vetor p foi aceite
p final:[
(163469052328813619960938398790292444337 29, 255420394263771281188966248109831944277, 3, 1375
457470007060800742714033668919527969)
(246650140975164683603736976255133205229 7, 308312676218955854504671220318916506537, 3, 55338

48191346211292827892470527044820 04)
(26509918359179440591569516030032467156030889738663679524432409985423049845608962256290365808226019497104321333787100526717102797128990655178694865393974872006786860800479922159552342898170573395675455235973477916251040700653028392327787125589008627853289194860421892632520452472722179302596186154420596764473683713667891 3, 456386142030968830373266851338520761167, 3, 10300813618790835140172031128790729105000637407997847962535968917355080797537169853335915094856633202185187595305121270906991370705716667841114505064251682587751780369619696508131118398225848033070603285641723654790269070629735201835886336183138834149979526418868382182307222005004481825362603589934352434883434098425190 4)
(46380683306958574555602537620256128764158159458609938984444125897500596956288396608066780232305666288383802822412932294198152959376191506483772615325765375901059877794399229307595744907705736445233618101383386694148302863306373290841879713145710365967614406738385729139419465393504664946383514152437204822507901156587483281476520332976244408281538759304595512387950219250784620944669624770891486864781115651471560561833242952602883948069046420411511337823123821144675967428231040192881386972799986002862420000918970453832412715322842792933569572372496583161369494276146498018266514128693591 3, 3265255635433058438502136945621813336929672297888628225718264053869691752342664111529614783621319206268267192588472108686721703840262497816301333405906970626071314709943753053997990766039608862811029165500883945685060937804338336833547671613941196038283393211417917444755295189362995414457585233300121344564708119400188366231560847951464133450430893998625944867124515592094218289563815146382410366399897693372867924693778095658597689878695599029942933442906222281996340478873903379989055248084126606942627201345849108836050222806927681823980655395124 1)
(23060699605857076360253260370108868811621429157182322591171047185932332513463354982617504347170984390729758435576083973727792165045761380715139994193059369023941686947191556973666871980700122615080221616799897413936862045807100332572993171022254424886863479455708475139579216508625411301779403470032430015685444273505663428162722982814086499478075794029128903372191111632974093883309762433111475211826341231528774910695425163615646561380964761760056263204047649697459143261310225970 1, 3, 997370981536175170674256675502129049692407361221116071386073720423024948031863514327698832819759493505243897439337669447365412533337784845621149305853606563672520365998986550739559324137720505838348892622509849130923692739631545876480339305031018097140167752774721983536700640068834973776507839749025532922728391483097584115874136808194660985373205745591842580421047671320892529678809799158863749931085988973710771046168126697298097235302008 34)
(49266203683031139270997339861625750947073, 19244610813684038777733335883447558963 7, 6622187158194944660142540145775777328304)
(35833617150369253930109795004212958926 57, 22396010718980783706318621877633099329 1, 28456051296770071323165675663142480130 9)
(60611644663333726708979453576122925533701204942470779653775476905058312674956651240420622169037441490438938142703131320418223974094458054315294444651053253266251770039237095311864494670627596885112576583875527306202775703877955370001656287918747667115013763915966093992462882263274766638570011436315682161933617611669629567199998349846751382543494198195445039559266533377, 55779016327280778710941461971712479404 7, 3, 284772284084226181211493855313472420354954594417872921349074184932071286942260354091625965137509037785714553264785868505222495355017290682825671919516699382769970348046746872387858003063993248821882250827922967453668283197911634831393919166501447049846568518751340100522052616911213981332830533404005427536903921030776052579827596024668566487180972538457135339876680313 85)
(21663585066780620849340428171385959605161657576547432861244111604895151363699576623671673890661724791361903311487951561009274398807361444090768624464239962182198055789730520567098454960163431643886356654416570104500470708110710407547244859352075222650645673884582767731653539663714971457684576774501932016314061412122881317443990926750173154909522912178014795930145914891, 6379624096903652897676316517110726633 27, 3, 485657491874900032234270712467221881050187980851801043219715182476766159892046204868192443644220450432986781132493022567575864114887054454055507823476838742091388066570773732111764343228538871139849572116330179890615536876315404598987578965285147656931888991078590675480865925526942677823837785381917434094915712670811149910986691719228649064920202832518842857586825880)
(63928518349942420812052395587579926646 09, 19977661984357006503766373621118727076 9, 3, 2051995010261495219124257747093870 45)
(20070207244887552008030068474330775178036434877407043289876551141113409348511088664089722232568345691118025786410392610030848566064871976366883622250900841427897896806817961005542197963459737115436091064673312074084470909191650970829089930533376271117181208993451103125195736376503610234922275246097662525167360090375841045476098513657862632970115137479635678276339406162384914857916831754102512010067653427 3, 3473818859644819213006593418183929783 71, 3, 1254431813057378867311837204454100974119180954868308790237921748248741419957595441506932102868427032198259804738225228938526936179720583443603671689105547216849917430747771058906631172260616493909464741080782801005463629766523007911647245243604529589001924611820520170944798564489097189854574974200758518660574980711632554420321773954318552565897468952341325700803755524237

98038090617199694062074666478948259)
(37209929270973273912374354115657950987 69, 23256205794358296195233971322286219367 3, 3163559982251886172018028269621412398028)
(11276375581808948383055924106909165547073, 17619336846576481848524881417045571167 3, 8336652193883426154961091191985080369676)
(62940334573236662353911792661512750503 9, 31470167286618331176955896330756375251 9, 7, 41222478257377118896410917907478844 1490)
(22319077986083940708819238476134803587 29, 27898847482604925886024048095168504484 1, 3, 1259614414820291376317776929306492823 00)
(36819231691091054200860521393789923537233177648950706073260616216189931455287642089772216734566741342286435105083703961981612642836554900105519448738683182470557557318192874711016959424920717375941949741611729969582917531460406327866447689129518820247415523594413129056672590340473991988360053794025913421663058351299161669370951273648299391644564261200850674232852491, 55514884902457867025283402365969914202 9, 3, 2866793564029750494269170013862410196416435930055477899339717816127327707072242969862309633521156843603456037028438452517078490226694469949915278762999866098819609031981790207497609741597431469674038076703479393806950478999534695465223505786375074153433400984729407752627379899562903809054461103553378375267570085555831571779822791086334165192514898482538244438017877)
(17735002582936500122786361708114932222 33, 22168753228670625153482952135143665277 9, 3, 1550049385601434827120863998475815236060)
(15131339678159231327086725966750324270 7457, 29553397808904748685716261653809227091 3, 3, 66763106334026044443806756503927211207547)
(35608107424230574185783651475137698952833, 27818833925180136082643477714951327306 9, 3, 880802152357043643829297276024357512413 0)
(11928352485987800699151739269703726868531 3, 23297563449194923240530740761140091540 1, 3, 36282023623922258943919688140382340009055)
(30037907860335056064279219369739972391883924099753740098584478949838341636599380525060915416472375772130640719492932582316794726185486344854990586441977855873901943941678100569593086227041732886330054090253442621050645620589089845452568167456497718490911127499672225991744387685826285630425535756821119032575504831668551690, 64640357890394687585124500992494244706 1, 2926971145583063168510596636088421151678849102411770791212947716262428493051622257145002539020582066549143976659404307266552217796411135752029260051418899200520263085810758341997038147435071559124101207355371632584304481306783237364451967581365357581558808272691302526000490778301656117850324522376292459159792752213943413)
(14633091952150294254807992706165013149 53, 18291364940187867818509990882706266436 9, 3, 1068362527766604794627380881282028283 74)
(16484808226066359511581408425260008333377, 25757512853228686736845950664468763020 9, 3, 399303990486701906098723137856015882606 0)
(58601875683895985857976292096443441582849, 22891357689021869475771989100173219368 3, 3, 3249035747374338911500505494997731339806 7)
(13246645471777758005295557684233744038553 7, 25872354437065933604092886102019031325 3, 3, 77441496906497343312468981274503240476261)
(14243262691972323441323054230855001884 73, 17804078364965404301653817788568752355 9, 3, 1294662811465442611814754523317290801688)
(16183907448405302840996563801425109660825 7, 31609194235166607111321413674658417306 3, 3, 16064511278898416593593809563504547601959 5)
(29330229026698669153894243321278426782849, 22914241427108335276479877594748770924 1, 3, 162638555796948175029322763231346012525 52)
(20436104098975257375166283674541527125766231111009862979572938373344812367379612060866342505995537032881186791917346360985774293833213539537699831449390358863268499161663143632362015462664775118081975706428207533185261634105284321115348194809514491711031609307511892559893430006283914246780175867218996043537377031316584035813641815466120867689267356478505260440477994071, 3, 729138512498745185670219707928586668518227016622172612686975704529893430663148329583051237670857038379857003438957321127210546368352154623843313096311854874148461910116358455162526897762373140120478575630491604043047512653353947126384462477837041443054018278988636269447560359022486921277880576885963542164668085102921920198661795965275383414264 68)
(15097686507592337381312689741355894109470932856720174697464684689342517369093996240694356091416307094111156346602344486679541635625090510699605761290764546258023075646943883921648378067254460821119373840009983887288396391034026041514772124623583685567979149664367537083258976790326470930855529123077242404849681476074325250892793259893483825967835551053806816731199961430243403182298766209733523350733838180781823365838299472799916095303054575548676628613375894277324 9, 650468954555024072640925805945181960581, 3, 14050017287940350604573359354567203259153553778115556768119700724616906274245083269463765832400157573633142264521277597263323400255241697592816617630644149033631280628489161856327646608822823029327677209876249567635177212279690251930539838411361080844109701379462557438028653272565067848197841827875508663371500540

564402490081594981126690464968648849540429049263731213197748651832135518106774132409293231907275740262030275597898012375619470202143786763520291985954727 0)
(413957772620513866799167438092457920014091, 32340450985977645843684956100973275001 1, 3, 2003532169790889329017911698906974760172 7)
(10141881860582749255549115318051889485 09, 25354704651456873138872788295129723712 7, 2, 3318624843125181401877377424678368788 37)
(51182709607265276360627342044623288176897, 19993245940337998578370055486180971944 1, 3, 4949419794759987192615689361495893052784 7)
(24893503058492543652652158155423806127 93, 31116878823115679565815197694279757659 9, 3, 8193488498752554161303023752799730970 98)
(44220000209668912948873742021607085593857, 17273437581901919120653805477190267810 1, 3, 3401410864966881623478445645889489607301 8)
(10223121739668643803165786899987669930480228784057928137715511204567143411374382512880437498792643403142955138475526018181936341143112370895850289633575626812082383077134591666370927378784354805753499026587961066307754362728267179275100939002241451600793839781946037580341376078141511114901125375034075276600497976894914496623829393630817124050426012680967514187247439026366898918541811232742510795161 7, 37108102974349871886863917604302256685 1, 3, 8161972078656655064028326267512106512166663200310355670607996700201220418818070975832139925188871818276712732756588631999222186130169238473853954011946160905069683000625648082928546048974968060185464781459221249789587272452650820314129953923582525808017548210891549385693363850428725052111033761108723535232534271334282167524311667312739576308057215403703921373897450632287096916005274095909358647233 9)
(21738011017898402987759893466476404409409, 33965642215466254668374833541369381889 7, 3, 132589487319279968110199008237576360035 92)
(60255337335840879644028181490552108724 17, 18829792917450274888758806715797533976 3, 3, 1390012757318498841808129683447748363069)
(41393182027837544923794914534179266142 7, 20696591013918772461897457267089633071 3, 2, 19748497869961740809782445109134146228 9)
(75953975552733217036625323622083632243 53, 23735617360229130323945413631901135076 1, 3, 3973732800589852327261525061096205362796)
(78596814712444697986084338109355196173847831039347219671334598459498057564586136137792567038078230142628719637348735156318384927763809238561843028671203695812260380202100223746373300859978266622912681274349607554415384236162592242070710375548057841778470375009241124684992771032221492476218007530684506381357028113277756260700199373169806540 9, 5730589664168997613999889141766132159261 3, 3871949993610487814694856758270797113957856679133695852741057287505118618609072416762414640139305905934660985880653271674229438735162655842488566319082499682090754909670206023916144688411410442598366009013727406882872039058755031990688732924299654072482535829571505169305534758525330531790435394210306246635800545771588651147586128984529858 88)
(47079780582140988615571415121328613955329, 18390539289898823677957584031768989826 3, 3, 110340705109911502507591773624210739001 1)
(15232805148782581190452917419163454950593, 23801258044972783110082683467442898360 3, 3, 461514239886972195334265683794871247441 2)
(10103211424265603135271236048046294750 29, 25258028560664007838178090120115736875 7, 2, 6342141547153284569086463706062276742 69)
(71855780066205040095034572795344156766977, 28068664088361343787122879998181311237 1, 3, 6780627036838557748167404514154326247697 0)
(10866467353360218857071620037813669857740 9, 21223569049531677455218007886354823940 9, 3, 21634775014307132420667531056132570196306)
(21004974854632323699809790037499622899393, 32820273210363005780952796933593160780 3, 3, 900198641016509621817150104197641475504 8)
(16325829730445596366013812533704367618497, 25509108953821244321896582083913074403 9, 3, 1155000468635517761049408510593159131119 1)
(12735772848209401687609738071601111592513, 19899645075327190136890215736876736863 3, 3, 7738999800138412671875475500944493935784)
(43518790883028244124270046989301991163777, 33999055377365815722085974210392180596 7, 3, 3672068327014255667030468959693568403005 7)
(25592636941503862864016065027991493458 33, 31990796176879828580020081284989366822 9, 3, 2111238474260446253849720470278093508820)
(69610540220001657238171513239067078106 9, 17402635055000414309542878309766769526 7, 2, 39264920814915319259991905679914569620 1)
(16295851740553934666883927686767740843059 3, 31827835430769403646257671263218243834 1, 3, 15057673180928640129896940898104293229221 3)
(62811384269256180617246769913812052312 7, 31405692134628090308623384956906026156 3, 5, 18364702894057895389863969970191984483 0)
(33105554598164073301753000107444338349 29, 20690971623852545813595625067152711468 3, 3, 1902456156784383632037532615642533577496)

(33674412408505568512827269916389767729782140392236734424734423868726611755998111669210192167368275503733960408644724098048204113841149700506776060290373360173201388532409536737947814679069256545533678911630328350106919234804104382409561374791880690974819685680735239092868707991616863733742850551875345458539874621366808892430033095024589492569925962001639312575165616310198568846811770522968561102595135897 7, 582848036817965428676000438729733196493, 3, 17291632516644195395445753442474658209754816001212694193373104989852577175233547563084431763829928378816342123051974789637390303699638843862223186457514602227499280901890336100686812955252398847635259478805937745495323354627666595536393439789385989811788116532562596542168292106885152039610138559363884751759791202281830861594234299916007330689799279577284724594655009175481150412946163204348738444931190770)
(12342823578745255123760908686885023138389, 308570589468631378094022717471255784597, 2, 1863932561705857799923742135588097830)
(54819344994734641657763569302355016645095656891490668137945641465915136686805319666658234850262779442279638830575368558064090336026504516437525040471951403812029949957215915954074673379888895656135002231474546166259507315624094160688089443570459852260825464546606444831793940675909617916440389063737157776155813791256016927065314552029577997957876517664886057726262302530544531099680055452439476119623588765076100253669853199492534831167610180620779531, 42547081786004332748374253897804430802 1, 3, 877026031929080397858155452074556724072765105237693339001766293352565453999342282525635574485018801466391621979962028930579509432093210882570728171049251126153194528558724954972533994947618767562811250673266026575712827284196729090712010790266450812046425412537221454133115145052357425656799911822256925103183420559113310379119106621144538581189897393189733651402489656373963267299344621056488568766820553529625602959977588677971582246205831708336 90)
(13240319807166001377050998551124794748364 9, 258599996233710964395527315451656147429, 3, 404917761775922286390082378105422932887 93)
(60684189645104023678790655718840008579 53, 189638092640950073996220799121375026811, 3, 10983870910748985095022422393722182360 56)
(30148745542478416660950442321323673739 69, 188429659640490104130940264508272960873, 3, 43180735079683951973717952657232483936 0)
(93791278530561233867740258106995977555058765937657688076688212721877659202327149631657431448500722982482438147110764736667984805894239919526834194710638597189341093658701149169100284044430312973857938134621798809704448211844134354789220286405409573175917767660224620121541213604098408244822851012975378187421732734813005339327468783027867077629768554764930032843751679981790794551872307124678300464932625795735826445893633, 5767372148170948481391287001320611344471, 3, 4721195520068756458186145274610476633502507459770520190820407239125582040391279386063386037827302917569602274291581266931296696963036909595054712298830064759091180791974333801008018177236889876180057277189989328301507854084981733180878628724270751916025884781004456410042646565502884709741791790517906165014391144407418519759963618312631254807765931415770419648799418029731070294643980034064461109331055770129923698270669 80)
(52511946162801203968443144372018994485 29, 32819966351750752480276965232511871553 3, 3, 3528731082352628596451395767527240456486)
(21416994806772124708652763022654414432193, 33464054385581444857269942222897522550 3, 3, 153687885140398103178521963287132994475 36)
(15111247253820809062194214982533514006788742296910459760809334044676449201107625617865716971284164361476512445686086131070066767438109251684543711796811066628696442689932222176754524000176689571023986221183622595904064857419078236889494002515508538138014578435350415097511561758228227521999372102130246493329403555990252541310096393125405369044462836121750676357270095335975068326511, 3, 1558663831205288716949136025995277111284293746822213793064350862113118376356295562077450875852974482853351846673238972400843910442227107533133287983119608854057865400689173462407666181699397974974566225407393726914110676842009496755916621925995790719451880552106338441121950022579610584895703286700433024861104063204155138050567746385879121864038805076 7)
(29755533713510007566391944249877457921 13, 185972085709437547289949651561734112007, 3, 17920495705632538973957784036152735490 39)
(39368846524059347295434485660346863034 09, 24605529077537092059646553537716789396 3, 3, 2876059742570633936235290758660961745441)
(88308430292390889382733944533685081026000092876835205659606567849690303389685158801837377414703575030634340340996090503727971607640013481351141192969826305267218431792185408192097390130014069298895155586864941529797841722801480509791086751610846408841757191425596067750784839471194839704842326302975056054821112136114602106242678270104738091363522158086602939176399543193628602433931485376229832004503947888979215730426902993045587789806690822780085535040068914060506038427213654340082183945552575805250274617799811759340197163700974126795087913888286862252111063203999562674314690981861083582360977409, 6543201440971287538768680921201621096373, 3, 6231183537517480229963532621482319671593654729744829212109707112751723390639469795605645301969193718152095177187347094002124803925777941655182387331314865087779334664260812524711172519934810177140201916378330886555557849040298547013571890220959681380978832480822564489109

81511008575577126467086664010422254650841814941846510544387633940832366472426630786217377049
44216812199375418806082213412177606930581942740088791241686231601820369031899565749891723805
81167574208202172417379366642471390776094416739004729884894735650958034376984335361414462080
30656108745187416460789684726107580520528927476589058761 3)
(29492071253202969545166241134044323226753, 23040680666564819957161125885972127520 9, 3, 1517
94505900049303719437518531239036779 65)
(74266370194698697907029917383129817986 9, 18566592548674674476757479345782454496 7, 2, 736914
36634575073778520218685210877888)
(13324474213765817882787149489916679352513, 20819490959009090441854921077994811488 3, 3, 1379
96703219684608581715411626849912681)
(22114376379152388125060025004193626678 97, 27642970473940485156325031255242033348 7, 3, 10688
95664996058093406267119364693501130)
(91595314371596365976444882627048437837313, 17889709838202415229774391138095398015 1, 3, 9880
63377087419351965400229286570114520)
(79544550935217667863620215701933949950209, 31072090209069401509226646758567949199 3, 3, 6750
07904444964841150094980264273777395 00)
(59970974718875660043963356942398337003414771154099216670877548291777008272642822344619045 12
57786095423072251335871896264213093144498710292773087100349226473908974047331845764942063848
14217006158211121870860880163909234991776155458580131176355368849297243010487683797880016972
21460241732766363612443701573952556640797415347789209677382491592598363173814273, 4971021716
15386350598190394162187221103, 3, 55081699875070005217196015280263224871101999559908382059 87
45790403060846215260583251691926245258817304650600046464712706151421876971346225063988708500
11758700119004410111324597601265491255152734239294713327171978856662596246467841972292418040
04923614375003048096760960084238368120422166481456304846149054642495431675174537507942700955
29970350956238495770 4)
(60391650580029450139397279628829794482 3, 30195825290014725069698639814414897241 1, 5, 637925
39802652570817522930041768414831)
(13773757327070185469194058002825135563 29, 17217196658837731836492572503531419454 1, 3, 36947
54026366420510115018585848291475858 1)
(43542424415986221021429624063062974474 73, 27214015259991388138393515039414359046 7, 3, 40016
73272752657881796404958809606146 43)
(34314521454554996235586061454639724859 69, 21446575909096872647241288409149828037 3, 3, 98923
92224181662084155792982100611125 21)
(85850321523462775655069969406086147769089, 33535281845102646740261706799252401472 3, 3, 8491
04607562897039395442996996409490632 17)
(15447360784354040545141760395877417168184500446827311534401211028724940297710224668750067905
51902295450079068732368184280232038992451612741244116976852797212788082245333142270423923884
04459921045779652220536989877824472569418604836348765782576286392800194448984356733346441590
62243139000363178497199284087092192044027829977581844365133300780270533839945828572450148744
69499269762790579884004355387467506056016034550465690069225455445887332614719454685079325475
97276027282218976726519701207166145523078086734643 3, 47749746152456332615943816000917796477 9
, 3, 11946642993078642531164997847228264706733611241163929481406980114319921650780415057022 4
60626301687234223489856312858248484514578646695407910606710823827019395343076543615261005937
86214409090971830446768605920863616145064898467548605536882114456132324328959507745946812813
25494884595444494856723438896341499118232305962501245200223225210452517853168250702258796042
99648409913231941410598677187036518067132493091881335502548692271094993962022901587774510547
99923266161077941175236370738673326732273166343565187 67)
(45650045536951013082229751559932388196 7, 22825022768475506541114875779966194098 3, 5, 184032
25887088895014266541651973799031 1)
(72352937202830371836386923482801515712257, 28262866094855613998588641985469342075 1, 3, 3443
90630833039693422664906151301416070 29)
(23350728285763617955943582495311296815 93, 29188410357204522444929478119139121019 9, 3, 79830
16684481389750157351388039431022 36)
(75098639600433189820966174274313297693953, 29335406093919214773814911825903631911 7, 3, 5638
29805805232580751044841243063277530 98)
(17147869697840714468799703439374284375617, 26793546402876116357499536624022319336 9, 3, 1675
11506840074416089206660826582024730 55)
(28098362539908354810105285490451925397889, 21951845734303402195394754289415566717 1, 3, 2655
29664981686483586531176345964649494 64)
(28653879515797890061919837755357056087 53, 17908674697373681288699898597098160054 7, 3, 23375
83319637908093264578659544330574658)
(73870837123560179082381345382784747942 9, 18467709280890044770595336345696186985 7, 2, 442678
85807117029681988338226411054661 2)
(53124045846686158865042611846727289761537, 20751580408861780806657270252627847563 1, 3, 3566
82539152303481779243391645520203155 35)
(21231788757046327144825906855557991717697, 33174669932884886163790479461809362058 9, 3, 1089

```
542284277601875246667798505492254785 8)
(48231230492147897713477245924680671057 13, 30144519057592436070923278702925419410 7, 3, 34632
56360699675657051520257047386535606)
(37596581548224837610098627581942761514369, 29372329334550654382889552798392782433 1, 3, 5855
28398185058031526032687510862589590 8)
(16971775496711756037812948388631453399091 3, 33147999017015148511353414821545807420 1, 3, 506
06141714402208060343361433547779427763)
(37630610699448212368154373047528586222 89, 23519131687155132730096483154705366389 3, 3, 14731
51441166052647709344666428013319901)
(14752377674842442192349278705262598813 37, 18440472093553052740436598381578248516 7, 3, 12248
47490603139366357090562047550934315)
(14233139408508340507907197386692354568 89, 17791424260635425634883996733365443211 1, 3, 80541
45333215517644402317133684311552 70)
(35324242341611102584705953505735346798 57, 22077651463506939115441220941084591749 1, 3, 22520
35537219811619569197413847556438666)
(34650530995500210055179713958525781431402291722555165035076811048471010786842600291331300117
90500939789596567222511946635587151431470040914386953565669854784876597062748435348697173269
03257303451055400433548486348929504073657320114889106796340406031555223916461781844746002175
87228983434800228928853997725759294020980687503 37, 364094411918235323009675669672989103539,
3, 73323205148244017953566333066585105355957394278458624946313734950285020128455365798696766
51758448643307758672919949605119971859275221513409018657367397877256032001809080325761980155
85807550022955013749770662162391285226604504267759661223482635240905189086166568685872751678
23049490588726899600442807598590897814525804764077 0)
(26425712732487025202451253304891939240 57, 33032140915608781503064066631114924050 7, 3, 20009
07075981018717623737058454440531525)
]
----------------------------------------------------------------------
b'S\x11\x94\x03\xc8\xbez\x14\x86q\x82.\xdb\xf1\xb2\x8a'
b'\xfc\x03b\xb9D\xf1\xab)\x9e<\x90f6r\xc6\xc8'
b"\xc7\x85\x18\xb6\xe3&\xb5y\xd0QW\x0c\xf1\xc4'\xc0"
b'\x8e\x1d6\x181\xa2g)\xda&\xcf\x85\xa1r\xbe\xb9'
b'\x91\x9dd\x8d\xaf2]\xfa\xb4\xf4\xfcq\xbb\xa3\xa4\xc9'
b'\x02\x89`\xca\x84\xb6\x18;;\xbe\xbeF\x87\xc5\xf3\xe6'
b'\xb9\xd3q @\xc7S!\x96\xb2\x0f\x13v>\xd2\xa4'
b'|[t\xf07\x85P\x88~JH\xe5B7L\xe8'
b'z{EC\x05I\xefEE\xeb\xa9d/s\xa4T'
b'\xf1\xddx\xd8\xc9\xdaW=\xf3\x8d\x03d\xb7\xfb\x995'
b'\x9e\xd05?4\x7f\x04\x13i\x99\xf8\x15\x8ey\x0c\xff'
b'\xc9\xfe\x01\xf5O\xc1g\x9f\x0f\xde\xd3\xa1\xed\x86\xde\xe3'
b'\xbe\n57\x01\xf5y\xbdm\xeb\xfb+\xb7e\x15\xd1'
b'\xe5\x0c\xce\xc9\xca\xa2\x87\xda\xbb\xb1p\xb4$73\x93'
b'\x88\xa3\xad\xb5\xefu\x7f\xd4d\xc5\r\x88\xd54\xf2R'
b'\x83\xfb\xc5\xef\xbf\xd0M3\xcd\x1b\x0f\x81\x18$\x8a\xd6'
b'\xd8\x19\xa9n\nH\xb5>H\xd0\xb2\x84O4\x05 '
b'\x94\xd0h\x11\x86\x0esA.\x84\x04\xb7\x9b\xdc\xebW'
b'\xac\xafI\x8cw\x11\x11\r\xa11\xbfN\x12\xf6t\x82'
b'\xc3\xa6\xa7\xddAj\xa7\x07\xbf\xcfx\t\x805\x0f,'
b'\xda\xc3\xe0\x08\xcb~\xe9\x84yu#\xee\x8fbz\xa9'
b'\xaa\xd3\xd3J\x9fa\x16r\x92\xaf\xf0\xbf\xb3Ys\x14'
b'liWI\xe7\x7f>\\\xe2\xf6\xd9 h\x89E\xd1'
b'\x8d*\xe0X9\xb0,\x8dI\xd4\x01\xc8\xdeQj*'
b'\x1cE\x97\xf5c\xee\x8e$\xf14\\Ul&&@'
b'\xf4\x83\x0f\xfdO\x8f\x9cpX=\x17\xbc\x06\xed\xde\x8c'
b'\x14a,\x009\xde\xdfYHY\x8b\x19\xe4\xa5\x16\x1d'
b"'A\x9ect^G\x98\xa5\x0c\x0b\xf8\x04\x00\n\x87"
b'\x0f\x10\x8aB\xa6\x1e\xc1\xf0(\x92Jh;\xb0E\xe3'
b'\x16c\xe1JH=\x7f\x07\xcfC\x022\x9d:\xab('
b'\xfb\xb4\x06Y\xd1\x837w\xac\xcf3\x96_\xf2\xa4J'
b'}vO\xf2\x9d\xdf\xaem\xa4\xdb\xa3\x96\xc0!jk'
b'!q4\r\x97\xb6\xca\xb1\xc8X3[\xb98\xef\x9d'
b'\x8a\xc4\xf0 \xd4\xdc5L\xe9&}H\xda\xc1\xc5\xd6'
b'\x00u\x93G\xd2\xa9\\\x18Al\xadK<\xf1\x05\x90'
b"\xad~\x07-n2\xaf\xb3\x1a\x8e\x07\xebE\xee\xaa'"
b'\xf7\x9a\x8a\x9a\x9d\x8b;\x84-\x17\x0e)OQ\x8dM'
b'\xde\x8a\xf2\xb3\x89\xc1\xa6\x1cA\x9f\xd5-qPpW'
b'\xe0\x08\xd1\xe2\x9d\x1a\xaf\xfc\x11?\x05\xca\x9dj\x82\xf9'
```

```
b'\x8c\xf9\xfa&6\x80F$\x80)\xa6t\x04v\x82,'
b"|VL'\xc06)\xac\x89\xd8/c~j\x06L"
b'\x93Y\xda*\x95;*"\x98\xcc\xb4\x85\xe4\xc8\xf0\x9c'
b'\x0bk\x08t\x0e\xfe\xfb\x03#Z(Uor\x06\xa4'
b'\x8ba\xe8\x1b\x06g\xc5\xa5\x90\x82\x98\xccD\xc1\x18^'
b'\x8c4w*\xee\x82\xa4\xea\xab\xfcP\xc7\xd9\x10Ev'
b'\xcb\x05e\xb7\x11\xdeG\x8d\r\x9d\xd1\xd6\xed\x16\x07\x06'
b'\x1b\x17|\xbd<;\xde\x99\xbc\xc4}\xfe\x81s\x08N'
b"\x0cj\x04\xf9O\x90\x9a\xd9\x85\xce\x18\xac?\x03\xce'"
b'\x91\rN\x93\x87?\x14l.\xba\xea\xa5\xd6,\xfe\xd5'
b'\x7f\xfa\xf1\xf4%\xedG~\x96\xdcSM\x90?\xb3\xee'
b"\xdfL\xcaG|+&\xd8\x07'\xdfP|\xbat:"
b'?:f\xd3\xfbZC\xfd.2\xe8RV\xcfz\x93'
b'eX\x0c:\x08\xc3\x9b\x189\x92g\xae\xb7\xae]X'
b'q\x18C\xc4\xa1\xca\xd5!\x82pj\x7f.\xd8\\b'
b'\xb6{Z\x1enA)\xf4\x8b9\x9b\xac\xb3\x05\x16V'
b'\x8e\x06\x16\xb8\xc5|\xde\xee\x0c\x0fc\xbe\x87}\xf5x'
b'\xea\xaf\xf2\x1e\xc4\x82\xe3c\xdd\x9dP\xba(g\xdaJ'
b'\x85\xcfp\xd1\x16\x0b\x16p\xba\x86\xd5\xed\xa1C\x18\xc8'
b'\xf79\x7f\xae\x12/\x88\tq\xb9\xcf\x1e}\xd1\xa3\xf4'
b'\xf6\xf9\x9eF/i\xa0\xf0\xffn\xe5\xe6r\x8d1\xbd'
b'\xe4\x9cr\xe1\xfe\x9e\xe4\x12\xdf\x7f/K\xb8\xb1\xee\x92'
b'\xa2\xa0\xe7\xb1?\xc3\xa9\x833\r\x8f\x9c\xb4\xdf\xb66'
b'\xa4.l6\xd0#\xfa\x14u\xfa?\x18\x8c;h\x9d'
b'~\t\tS*\t\x7f\x05\x11\x92@QBv\xba\x07'
b'ZH\xb6\xc1S\x96\xd7G\xc9\xff\xbfK\x8b\xab\x91\xe2'
b'\xc3\xa8\xaf\x02\xbb\x1077\xd9\xff\xa9.@\xdd\xaf\xa5'
b'\r!\xbb&\x823\xa1w\x8d\xcc\x17X\xb8\xfc:S'
b'*\x97TB\x9a\xfb^XHC\x08;S\x0b\xe7\x1b'
b'\xe4\x96\x8f\xe6\xf3f\x8d\xaa(6\x10]\xf8\xbb\xc7\xa6'
b'd\x9e\x0b@f>\xa1\x0b\xa1\xbc\xa8\xfcs\xe5N='
b'\xc8\x1aP@\xd0\xb1\xf0\xb6\xe4\x1e\xb7n\xeb\xbf3\xcd'
b'\x9dD\xc8a\xbb\x9d\t\xea@&71i\x9e%\xe1'
b'v\xe7\xbb\n\xe7\x8f\x0c\x80\xc1w\xb4C\x83@k\xa0'
b'#\x89$\xe1\xbc\xe4\xb87\xff:\x8c\xe9p\xaa\xf5\x99'
b'\xdd7\xb1k\xa2\x87\xbe\xaf\x01,\x87q\x1c\xec\x14 '
b'~\x86\xce\x00J\xcd9\xae\xe7^W\x1b\xb5\tt '
b'j\x04\x98\xe5b\x97X\x0e\x1bp\xbf8\x8f\x00\x94\xb3'
b'<\xfd\xa0\x1a\xa4\xaes\xd3\xda\x168Mt\xf0\xc4B'
b'\xb7]\x19\xaa.\xa6\x0c@<\r0\x98\xbc\xdf\x96\x1a'
b"\xc9\x1a\xb9\x05X$\x08\x89\x13k\\hs0'\x1d"
b'n\xd3\x84i\xc6)\x92\xb7(\xee\x19%\x06\x98fs'
b'\xb5\xf7\xb7:h\xcf\x1d!\x15\x94JN@\xbc\xbf9'
b'\xea %\xf8\xe5\x1a\x0f\x15\xa0\x81L\xcf\xd8\xe8\xcb\x11'
b'\xbe#U[\x8f\xaf\xdf\xfe\xcap\x01w\x0c\x87x\xfb'
b'\xe7\xfe\xa3\\5?\xc9e\x90\x93\xf5x\xf1\xba\x98}'
b'\x8d\xca\\\x0b8d&\x7f0}\x07\xb4\t\x0e\xa0\x04'
b'Z\xf1\x01\x86\xdce\xaa\x17<{\x97\xf4\xeaa\xadX'
b'SQ\xd1\xb7\xb8\x8f\xb7\xfa:\t\x0f\x16\x19\xe8\x1e\x0b'
b'\xe7\xbe\xa7\xc9J\xab\xba\xb1:\x81vM\xc26\x8e\xf7'
b'\xccd\x8b\xae\xdaFLp}{\xe3\x89\x0f+\xc6\x84'
b'5\x01\x83\x9f\xe9M\xb0\xe9\x8ax\x9d\xe2\xcf\x18\x1b\xf3'
b'\x96\xb6\xb0\x14\x9e*\xad\x1f\x8e\x14\xbe3&\x9fn\x1f'
b'\x85\x0e\x19F\x1a\xe5\xb8\xd3\xf5\xcf\x96\x8e\xc5\x99\xf7~'
b'H\xa0<\x14\x93d``O\x1c\x0e\x10\xe9\x87uX'
b'\x10\x94\xd04[}\xb4A\xd9\x03\xbe\xdea8\xb5\xd3'
b'\xba_\xed\x06X\xeaz\xaes+V\x80\n\x02\xec\xd5'
b'\xa1\xfa\x9d\x92\xda-E\xa0e\xa77\x0f\x9abd\xda'
b'j\x0e)`\xa2\x86\x0c\xe9g0\xde\x02\xf1"\xb5\xb3'
b'\x10\x01\xa2\xc8\x18l\x85gCC\x9a\xe9\xd6F@\xb0'
b'K\xd60\xcaR!\xfc\x8e|%P~Az=/'
```

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 [28]:
```
def dec_fujisaki_OT(sk,pk,y,c,tau):
    r = int(dec(sk,pk,c))
    rlinha = h_OT(r,y,tau)
    if c != f_p_OT(pk,r,int.from_bytes(rlinha,"big")):
        print("má decifração")
        return None
    else:
        print("boa decifração")
        res = bytes(a ^^ b for a, b in zip(y, g(r)))
        return res
```

In [29]:
```
# Decifrar e verificar as mensagens
decrypted_messages = []
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]}")
        print("len(ciphertext_vector): ",len(ciphertext_vector))
        # 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 = dec_fujisaki_OT(sk, receiver.p[idx], y, c, 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].encode('utf-8'):
                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.")
```

```
Número da mensagem: 1
Índice no vetor ciphertext_vector: 2
Chave privada (sk): 39822229966415039273058785286330073985
Chave pública (pk): (26509918359179440591569516030032467156030889738663679524432409985423049
84560896225629036580822601949710432133378710052671710279712899065517869486539397487200678686
08004799221595523428981705733956754552359734779162510407006530283923277871255890086278532891
94860421892632520452472722179302596186154420596764473683713667891 3, 456386142030968830373266
851338520761167, 3, 10300813618790835140172031128790729105000637407997847962535968917355080 7
97537169853335915094856633202185187595305121270906991370705716667841114505064251682587751780
36961969650813111839822584803307060328564172365479026907062973520183588633618313883414997952
64188683821823072220050044818253626035899343524348834340984251904)
len(ciphertext_vector):  100
boa decifração
Mensagem decifrada: b'mensagem2'
Decifração bem-sucedida! A mensagem decifrada corresponde à original.

Número da mensagem: 2
Índice no vetor ciphertext_vector: 3
Chave privada (sk): 343347144649756448864458167347865142323
Chave pública (pk): (46380683306958574555602537620256128764158159458609938984444125897500596
95628839660806678023230566628838380282241293229419815295937619150648377261532576537590105987
77943992293075957449077057364452336181013833866941483028633063732908418797131457103659676144
06738385729139419465393504664946383514152437204822507901156587483281476520332976244408281538
75930459551238795021925078462094466962477089148686478111565147156056183324295260288394806904
64204115113378231238211446759674282310401928813869727999860028624200009189704538324127153228
4279293356957237249, 65831613694942761464980182665141286935 9, 3, 32652556354330584385021369 4
56218133692967229788862822571826405386969175234266411152961478362131920626826719258847210868
67217038402624978163013334059069706260713147099437530539979907660396088628110291655008839456
85060937804338336833547671613941196038283393211417917444755295189362995414457585233300121344
56470811940018836623156084795146413345043089399862594486712451559209421828956381514638241036
63998976933728679246937780956585976898786955990299429334429062222819963404788739033799890552
48084126606942627201345849108836050222806927681823980655395 1241)
len(ciphertext_vector):  100
boa decifração
Mensagem decifrada: b'mensagem3'
Decifração bem-sucedida! A mensagem decifrada corresponde à original.

Número da mensagem: 3
Índice no vetor ciphertext_vector: 4
Chave privada (sk): 353368548385613739706645266031680521474
Chave pública (pk): (23060699605857076360253260370108868811621429157182322591171047185932332
51346335498261750434717098439072975843557608397372779216504576138071513999419305936902394168
69471915569736668719807001226150802216167998974139368620458071003325729931710222544248868634
79455708475139579216508625411301779403470032430015685444273505663428162722982814086499478075
79402912890337219111163297409388330976243311147521182634123152877491069542516361564656138096
47617, 60056263204047649697459143261310225970 1, 3, 99737098153617517067425667550212904969240
73612211160713860737204230249480318635143276988328197594935052438974393376694473654125333377
84845621149305853606563672520365998986550739559324137720505838348892622509849130923692739631
54587648033930503101809714016775277472198353670064006883497377650783974902553292272839148309
75841158741368081946609853732057455918425804210476713208925296788097991588637499310859889737
10771046168126697298097235302008 34)
len(ciphertext_vector):  100
boa decifração
Mensagem decifrada: b'mensagem4'
Decifração bem-sucedida! A mensagem decifrada corresponde à original.

Número da mensagem: 4
Índice no vetor ciphertext_vector: 7
Chave privada (sk): 25555781192704580403853278163118677 8513
Chave pública (pk): (60611644663333726708979453576122925533701204942470779653775476905058312
67495665124042062216903744149043893814270313132041822397409445805431529444465105325326625177
00392370953118644946706275968851125765838755273062027757038779553700016562879187476671150137
63915966093992462882263274766638570011436315682161933617611669629567199998349846751382543494
19819544503955926653337 7, 55779016327280778710941461971712479404 7, 3, 2847722840842261812114
93855313472420354954594417872921349074184932071286942260354091625965137509037785714553264785
86850522249535501729068282567191951669938276997034804674687238785800306399324882188225082792
```

```
29674536682831979116348313939191665014470498465685187513401005220526169112139813328305334040
05427536903921030776052579827596024668566487180972538457135339876680313851)
len(ciphertext_vector):  100
boa decifração
Mensagem decifrada: b'mensagem7'
Decifração bem-sucedida! A mensagem decifrada corresponde à original.

Número da mensagem: 5
Índice no vetor ciphertext_vector: 8
Chave privada (sk): 19834113053857732560386088203989210004 3
Chave pública (pk): (21663585066780620849340428171385959605161657576547432861244111604895151
36369957662367167389066172479136190331148795156100927439880736144409076862446423996218219805
57897305205670984549601634316438863566544165701045004707081107104075472448593520752226506456
73884582767731653539663714971457684576774501932016314061412122881317443990926750173154909522
91217801479593014591489, 63796240969036528976763165171107266332 7, 3, 48565749187490003223427
07124672218810501879808518010432197151824767661598920462048681924436442204504329867811324930
22567575864114887054454055507823476838742091388066570773732111764343228538871139849572116330
17989061553687631540459898757896528514765693188899107859067548086592552694267782383778538191
74340949157126708111499109866917192286490649202028325188428575868258 0)
len(ciphertext_vector):  100
boa decifração
Mensagem decifrada: b'mensagem8'
Decifração bem-sucedida! A mensagem decifrada corresponde à original.

Número da mensagem: 6
Índice no vetor ciphertext_vector: 10
Chave privada (sk): 64469076890246090008274633860416453612
Chave pública (pk): (20070207244887552008030068474330775178036434877407043289876551141113409
34851108866408972223256834569111802578641039261003084856606487197636688362225090084142789789
68068179610055421979634597371154360910646733120740844709091916509708290899305333762711171812
08993451103125195736376503610234922275246097662525167360090375841045476098513657862632970115
13747963567827633940616238491485791683175410251201006765342 73, 34738188596448192130065934181
8392978371, 3, 12544318130573788673118372044541009741191809548683087902379217482487414199575
95441506932102868427032198259804738225228938526936179720583443603671689105547216849917430747
77105890663117226061649390946474108078280100546362976652300791164724524360452958900192461182
05201709447985644890971898545749742007585186605749807116325544203217739543185525658974689523
41325700803755524237980380906171996940620746666478948259)
len(ciphertext_vector):  100
boa decifração
Mensagem decifrada: b'mensagem10'
Decifração bem-sucedida! A mensagem decifrada corresponde à original.

Número da mensagem: 7
Índice no vetor ciphertext_vector: 15
Chave privada (sk): 70317692219282548406715057325397986923
Chave pública (pk): (36819231691091054200860521393789923537233177648950706073260616216189931
45528764208977221673456674134228643510508370396198161264283655490010551944873868318247055755
73181928747110169594249207173759419497416117299695829175314604063278664476891295188202474155
23594413129056672590340473991988360053794025913421663058351299161669370951273648299391644564
26120085067423285249, 55514884902457867025283402365969914202 9, 3, 28667935640297504942691700
13862410196416435930055477899339717816127327707072242969862309633521156843603456037028438452
51707849022669446994991527876299986609881960903198179020749760974159743146967403807670347939
38069504789995346954652235057863750741534334009847294077526273798995629038090544611035533783
75267570085555831571779822791086334165192514898482538244438017877)
len(ciphertext_vector):  100
boa decifração
Mensagem decifrada: b'mensagem15'
Decifração bem-sucedida! A mensagem decifrada corresponde à original.

Número da mensagem: 8
Índice no vetor ciphertext_vector: 20
Chave privada (sk): 49348377498292718791944428579980742 1619
Chave pública (pk): (30037907860335056064279219369739972391883924099753740098584478949838341
63659938052506091541647237577213064071949293258231679472618548634485499058644197785587390194
39416781005695930862270417328863300540902534426210506456205890898454525681674564977184909111
27499672225991744387685826285630425535756821119032575504831668 55169, 64640357890394687585124
```

```
5009924942437061, 3, 2926971145583063168510596636088421151678849102411770791212947716262428493051622257145002539020582066549143976659404307266552217796411135752029260051418899200520263085810758341997038147435071559124101207355371632584304481306783237364451967581365357581558808272691302526000490778301656117850324522376292459159792752213943413)
len(ciphertext_vector):  100
boa decifração
Mensagem decifrada: b'mensagem20'
Decifração bem-sucedida! A mensagem decifrada corresponde à original.

Número da mensagem: 9
Índice no vetor ciphertext_vector: 28
Chave privada (sk): 74395228704494580996344400193131340377
Chave pública (pk): (204361040989752573751662836745415271257662311110098629795729383733448123673796120608663425059955370328811867919173463609857742938332135395376998314493903588632684991616631436323620154626647751180819757064282075331852616341052843211153481948095144917110316093075118925598934300062839142467801758672189960435373770313165840358136418154661208676892673, 568398012504521375478505260440477994071, 3, 729138512498745185670219707928586668518227016622172612686975704529893430663148329583051237670857038379857003438957321127210546368352154623843313096311854874148461910116358455162526897762373140120478575630491604043047512653353947126384462477837041443054018278988636269447560359022486921277880576885963542164668085102921920198661795965275383414264 68)
len(ciphertext_vector):  100
boa decifração
Mensagem decifrada: b'mensagem28'
Decifração bem-sucedida! A mensagem decifrada corresponde à original.

Número da mensagem: 10
Índice no vetor ciphertext_vector: 29
Chave privada (sk): 57531422204828047005062251041721297926
Chave pública (pk): (150976865075923373813126897413558941094709328567201746974646846893425173690939962406943560914163070941111563466023444866795416356250905106996057612907645462580230756469438839216483780672544608211193738400099838872883963910340260415147721246235836855679791496643675370832589767903264709308555291230772424084896814760743252508927932589348382596783555105380368173119996143024340318229876620973352335073383818078182336583829947279991609530305457554867662861337589427732 49, 650468954555024072640925805945181960581, 3, 14050017287940350604573359354567203259153553778115556768119700724616906274245083269463765832400157573633142264521277597263323400255241697592816617630644149033631280628489161856327646608822823029327677209876249567635177212279690251930539838411361080844109701379462557438028653272565067848197841827875508663371500545644024900815949811266904649686488495404290492637312131977486518321355181086774132409293231907275740262030275597898012375619470202143786763520291985954727 0)
len(ciphertext_vector):  100
boa decifração
Mensagem decifrada: b'mensagem29'
Decifração bem-sucedida! A mensagem decifrada corresponde à original.

Número da mensagem: 11
Índice no vetor ciphertext_vector: 35
Chave privada (sk): 31384077101601614377036807813818324 8127
Chave pública (pk): (102231217396686438031657868999876699304802287840579281377155112045671434113743825128804374987926434031429551384755260181819363411431123708958502896335756268120823830771345916663709273787843548057534990265879610663077543627282671792751009390022414516007938397819460375803413760781415111149011253750340752766004979768949144966238293936308171240504260126809675141872474390263668989185418112327425107951617, 3710810297434987188686391760430225566851, 3, 8161972078656655064028326267512106512166663200310355670607996700201220418818070975832139925188871818276712732756588631999222186130169238473853954011946160905069683000625648082928546048974968060185464781459221249789587272452650820314129953923582525808017548210891549385693363850428725052111033761108723535232534271334282167524311667312739576308057215403703921373897450632287096916005274095909358647233 9)
len(ciphertext_vector):  100
boa decifração
Mensagem decifrada: b'mensagem35'
Decifração bem-sucedida! A mensagem decifrada corresponde à original.

Número da mensagem: 12
Índice no vetor ciphertext_vector: 40
Chave privada (sk): 14723337701587740413986877378953880 1097
```

```
Chave pública (pk): (785968147124446979860843381093551961738478310393472196713345984594980575645861361377925670380782301426287196373487351563183849277638092385618430286712036958122603802021002237463733008599782666229126812743496075544153842361625922420707103755480578417784703750092411246849927710322214924762180075306845063813570281132777562607001993731698065409, 57305896641689976139998891476613215926 1, 3, 3871949993610487814694856758270797113957856679133695852741057287505118618609072416762414640139305905934660985880653271674229438735162655842488566319082499682090754909670206023916144688411410442598366009013727406882872039058755031990688732924299654072482535829571505169305534758525330531790435394210306246635800545771588651147586128984529857150516930553475852533053179043539421030624663580054577158865114758612898452985888)
len(ciphertext_vector):  100
boa decifração
Mensagem decifrada: b'mensagem40'
Decifração bem-sucedida! A mensagem decifrada corresponde à original.

Número da mensagem: 13
Índice no vetor ciphertext_vector: 55
Chave privada (sk): 24369066587595607576662159599 1091130121
Chave pública (pk): (336744124085055685128272699163897677297821403922367344247344238687266117559981116692101921673682755037339604086447240980482041138411497005067760602903733601732013885324095367379478146790692565455336789116303283501069192348041043824095613747918806909748196856807352390928687079916168637337428505518753454585398746213668088924300330950245894925699259620016393125751656161631019856884681177052296856110259513589779733196493, 3, 17291632516644195395445753442474658209754816001212694193373104989852577175233547563084431763829928378816342123051974789637390303699638843862223186457514602227499280901890336100686812955252398847635259478805937745495323354627666595536393439789385898811788116532562596542168292106885152039610138559363884751759791202281830861594234299916007330689799279577284724594655009175481150412946163204348738444931190770)
len(ciphertext_vector):  100
boa decifração
Mensagem decifrada: b'mensagem55'
Decifração bem-sucedida! A mensagem decifrada corresponde à original.

Número da mensagem: 14
Índice no vetor ciphertext_vector: 57
Chave privada (sk): 28873741311053320790081142732738716 6632
Chave pública (pk): (548193449947346416577635693023550166450956568914906681379456414659151366868053196666582348502627794422796388305753685580640903360265045164375250404719514038120299499572159159540746733798888956561350022314745461662595073156240941606880894435704598522608254645466064448317939406759096179164403890637371577761558137912560169270653145520295779979578765176648860577262623025305445310996800554524394761196235887650761002536698531994925348311676101806207795 3, 42547081786004332748374253897804430802 1, 3, 8770260319290803978585155452074556724072765105237693339001766293352565453999342282525635574485018801466391621979962028930579509432093210882570728171049251126153194528558724954972533994947618767562811250673266026575712827284196729090712010790266450812046425412537221454133115145052357425656799911822256925103183420559113310379119106621144538581189897393189733651402489656373963267299344621056488568766820553529625602959977588677971582246205831708336 90)
len(ciphertext_vector):  100
boa decifração
Mensagem decifrada: b'mensagem57'
Decifração bem-sucedida! A mensagem decifrada corresponde à original.

Número da mensagem: 15
Índice no vetor ciphertext_vector: 61
Chave privada (sk): 6584249937699196770380055949830641 5128
Chave pública (pk): (937912785305612338677402581069959775550587659376576880766882127218776592023271496316574314485007229824824381471107647366679848058942399195268341947106385971893410936587011491691002840444303129738579381346217988097044482118441343547892202864054095731759177676602246201215412136040984082448228510129753781874217327348130053393274687830278670776297685547649300328437516799817907945518723071246783004649326257957358264458936335767372148170948481391287001320611344713, 4721195520068756458186145274610476633502507459770520190820407239125582040391279386063386037827302917569602274291581266931296696963036909595054712298830064759091180791974333801008018177236889876180057277189989328301507854084981733180878628724270751916025884781004456410042646565502884709741791790517901650143911444074185197599636183126312548077659314157704196487994180297310702946439800340644611093310557701299236982706698 0)
len(ciphertext_vector):  100
boa decifração
```

Mensagem decifrada: b'mensagem61'
Decifração bem-sucedida! A mensagem decifrada corresponde à original.

Número da mensagem: 16
Índice no vetor ciphertext_vector: 64
Chave privada (sk): 17735521666414614521055034030121319328 2
Chave pública (pk): (151112472538208090621942149825335140067887422969104597608093340446764492011076256178657169712841643614765124456860861310700667674381092516845437117968110666286964426899322221767545240001766895710239862211836225959040648574190782368894940025155085381380145784353504150975115617582282275219993721021302464933294035559990252541310096393125405369044462836121750103129795067635727009533597506832651 1, 155866383120528871694913602599527711128429374682221379306435086211311837635629556207745087585297448285335184667323897240084391044222710753313328798311960885405786540068917346240766618169939797497456622540739372691411067684200949675591662192599579071945188055210633844112195002257961058489570328670043302486110406320415513805056774638587912186403880507671)
len(ciphertext_vector): 100
boa decifração
Mensagem decifrada: b'mensagem64'
Decifração bem-sucedida! A mensagem decifrada corresponde à original.

Número da mensagem: 17
Índice no vetor ciphertext_vector: 67
Chave privada (sk): 62018927454818381985298315087734812360 4
Chave pública (pk): (883084302923908893827339445336850810260000928768352056596065678496903033896851588018373774147035750306343403409960905037279716076400134813511411929698263052672184317921854081920973901300140692988951555868649415297978417228014805097910867516108464088417571914255960677507848394711948397048423263029750560548211121361146021062426782701047380913635221580866029391763995431936286024339314853762298320045039478889792157304269029930455877898906690822780085535040068914060506038427213654340082183945552575805250274617799811759340197163700974126795087913888286862252111063203999562674314690981861083582360977409, 654320144097128753876868092120162109637, 3, 623118353751748022996353262148231967159365472974482921210970711275172339063946979560564530196919371815209517718734709400212480392577794165518238733131486508779334664260812524711172519934810177140201916378330886555557849040298547013571890220959681380978832480822564489108151100857557712646708666401042225465084181494184651054438763394083236647242663078621737704944216812199375418806082213412177606930581942740088791241686231601820369031899565749891723805811675742082021724173793666424713907760944167390047298848947356509580343769843353614144620803065610874518741646078968472610758052052892747658905876 13)
len(ciphertext_vector): 100
boa decifração
Mensagem decifrada: b'mensagem67'
Decifração bem-sucedida! A mensagem decifrada corresponde à original.

Número da mensagem: 18
Índice no vetor ciphertext_vector: 74
Chave privada (sk): 33802175906838244659207326741216129153 9
Chave pública (pk): (599709747188756600439633569423983370034147711540992166708775482917770082726428223446190451257786095423072251335871896264213093144498710292773087100349226473908974047331845764942063848142170061582111218708608801639092349917761554585801311763553688492972430104876837978800169722146024173276636361244370157395255664079741534778920967738249159259836317381427 3, 4971021716153863505981903941621872211 03, 5508169987507000521719601528026322487110199955990838205987457904030608462152605832516919262452588173046506000464647127061514218769713462250639887085001175870011900441011132459760126549125515273423929471332717197885666259624646784197229241804004923614375003048096760960084238368120422166481456304846149054642495431675174537507942700955299703509562384957704)
len(ciphertext_vector): 100
boa decifração
Mensagem decifrada: b'mensagem74'
Decifração bem-sucedida! A mensagem decifrada corresponde à original.

Número da mensagem: 19
Índice no vetor ciphertext_vector: 80
Chave privada (sk): 25028568046153054179958781942774875344 2
Chave pública (pk): (154473607843540405451417603958774171681845004468273115344012110287249402977102246687500679055190229545007906873236818428023203899245161274124411697685279721278808224533314227042392388404599210457796522205369898778244725694186048363487657825762863928001944489843567333464415906224313900036317849719928408709219204402782997758184436513330078027053385 1

```
3994582857245014874469499269762790579884004355387467506056016034550465690069225455445887332614719454685079325475972760272822189767265197012071661455230780867346433, 4774974615245633261594381600091779647791, 3, 1194664299307864253116499784722826470673361124116392948140698011431992165078041505702246062630168723422348985631285824848451457864669540791060671082382701939534307654361526100593786214409090971830446768605920863616145064898467548605536882114456132324328959507745946812813254948845954444948567234388963414991182323059625012452002232252104525178531682507022587960429964840991323194141059867718703651806713249309188133550254869227109499396202290158777451054799923266161077941175236370738673326732273166343565187671}
len(ciphertext_vector):  100
boa decifração
Mensagem decifrada: b'mensagem80'
Decifração bem-sucedida! A mensagem decifrada corresponde à original.

Número da mensagem: 20
Índice no vetor ciphertext_vector: 98
Chave privada (sk): 32640409292152014032054013345466368468 4
Chave pública (pk): (3465053099550021005517971395852578143140229172255516503507681104847101078684260029133130011790500939789596567222511946635587151431470040914386953565669854784876597062748435348697173269032573034510554004335484863489295040736573201148891067963404060315552239164617818447460021758722898343480022892885399772575929402098068750337, 36409441191823532300967566967298910353 9, 3, 7332320514824401795356633306658510535595739427845862494631373495028502012845536579869676651758448643307758672919949605119971859275221513409018657367397877256032001809080325761980155858075500229550137497706621623912852266045042677596612234826352409051890861665686858727516782304949058872689960044280759859089781452580476407 70)
len(ciphertext_vector):  100
boa decifração
Mensagem decifrada: b'mensagem98'
Decifração bem-sucedida! A mensagem decifrada corresponde à original.
```