TP3-Problema1-Dilithium

May 30, 2022

1 TRABALHO PRÁTICO 3 - GRUPO 14

1.1 Exercício 1 - Problema 1 - DILITHIUM

Este problema consistia em implementar a técnica **DILITHIUM** que é um esquema de assinatura digital presente no concurso NIST PQC e usa o esquema LWE básico como ponto de partida. Nesta implementação do Dilithium foi utilizado os passos presentados no paper Dilithium

1.1.1 IMPORTS

[10]:
```
from math import *
from cryptography.hazmat.primitives import hashes
from pickle import load, dumps
import random as rn
import os, sys
import numpy as np
```

1.1.2 RESOLUÇÃO DO PROBLEMA

Na classe abaixo é implementado o algoritmo DILITHIUM que irá gerar uma assinatura utilizando uma chave privada, sendo a chave pública utilizada para verificar a autenbticidade da assinatura. Assim sendo, será necessário implementar 3 funcionalidades principais:

Geração do par de chaves: A função `Gen` tem como objetivo gerar o par de chaves a ser utilizado para a assinatura da mensagem e para a verificação da assinatura. Para isso começamos por gerar um valor aleatório de 32 bytes, ζ, que será utilizado pela função de hash `tripleH` para gerar as variáveis ρ, ς e K. A variável ς será utilizada na função `vectorGen` para gerar os vetores $\mathbf{s}_1$ e $\mathbf{s}_2$ $\in S_\eta^l \times S_\eta^k$, já a variavél ρ será utilizada pela função `ExpandA` para gerar a matriz $\mathbf{A} \in R_q^{k \times l}$.

- **Geração das Samples $\mathbf{s}_1$ e $\mathbf{s}_2$:** Estes dois vetores serão gerados apartir da condição $||w|| < \tau, w \in \{\mathbf{s}_1, \mathbf{s}_2\}$, sendo que para isso é necessário que todos os elementos x de $\mathbf{s}_1$ e $\mathbf{s}_2$ seja gerados segundo a condição $x = ||n||_\infty = |n \ mod^{\pm} \ q|, n \in \varsigma$.
- **Geração da matriz A:** Para gerarmos a matriz A começamos por gerar um inteiro de 2 bytes em formato *little endian* apartir da expressão $n * i + j, i \in \{0...k\}$ e $j \in \{0...l\}$. Estes 2 bytes vão ser utilizados juntamente coma a variável ρ na função de hash `genHash128`, sendo o output utilizado pela função `genInteger`. Esta função começa por utilizar os 3 primeiros bytes do output recebido, transformando o bit mais alto do terceiro byte a zero. Este 3 bytes

serão interpretados como um inteiro em formato *little endian*, gerando-se assim os coeficentes dos polinómios da matriz A.

De seguida geramos o vetor t através da expressão $t = A * \mathbf{s}_1 + \mathbf{s}_2$ que será utilizado na função `Power2Round` que tem como objetivo partir *bit wise* os elementos de R_q devolvendo assim o par (t_1, t_0). A variável t_1 será utilizada na função de hash `CRH` juntamento com a variável ρ para gerar a variável tr. No final a chave pública é dada por $pk = (\rho, t_1)$, já a chave privada será dada por $sk = (\rho, K, tr, \mathbf{s}_1, \mathbf{s}_2, t_0)$

Geração da assinatura: A função `Sign` tem como objetivo assinar uma mensagem a ser enviada, devolvendo a assinatura gerada. Para isso, utilizamos a chave privada sk e a mensagem em bytes M. Iniciamos a função com a geração da matriz A utilizando a função *ExpandA* e a variavél ρ. De seguida, geramos a variável μ através da função *CRH* utilizando as variáveis tr e M e a variável ρ' através da função *CRH* e das variáveis K e μ. Ao longo de um ciclo while vamos gerando as variáveis z e h de forma a que estam respeitem a condição: $||\mathbf{z}||_\infty < \gamma_1 - \beta$ **and** $||\mathbf{r}_0||_\infty < \gamma_2 - \beta$ **and** $||c * \mathbf{t}_0||_\infty < \gamma_2$ **and** # of 1's in $\mathbf{h} \leq \omega$. Para isso começamos por gerar a variável y através da função `ExpandMask` utilizando as variáveis ρ' e k, com k inicialmente igual a 0.

- **Geração do vetor y:** Para gerarmos o vetor y começamos por gerar um inteiro de 2 bytes em formato *little endian* apartir da expressão $k+i, i \in \{0...l\}$. Estes 2 bytes vão ser utilizados juntamente coma a variável ρ' na função de hash `genHash256`, sendo o output utilizado pela função `genIntegerMask`. Esta função começa por utilizar os 3 primeiros bytes do output recebido, transformando os dois 2 bits ou os 4 bits do terceiro byte dependendo se o γ_1 é igual a 2^{17} ou 2^{19}, respetivamente. Este 3 bytes serão interpretados como um inteiro em formato *little endian*, gerando-se assim os coeficentes dos polinómios do vetor y.

De seguida, geramos o vetor w através da expressão $w = A * y$ sendo este vetor utilizado na função `HighBits` que tem como objetivo extrair os *high-order bits* do output da função `Decompose`. Esta última tem como objetivo reduzir o tamanho da chave pública através da geração dos *higher-order* and *lower-order bits* dos elementos pertencentes a Z_q. Assim, geramos a variável $\mathbf{w}_1$ que será utilizada, juntamente com a variável μ, na função de hash `H` para assim calcular a variável $\tilde{c}$. O output desta função será utilizado pela função `SampleInBall` para gerar o vetor c.

- **Geração do vetor c:** Para gerarmos o vetor c começamos por utilizar a função `genHash256` para assim gerar uma hash de n bytes apartir da variável recebida como parâmetro. De seguida, geramos os *sign bits* que podem ser obtidos utilizando a função `genSignBits`. Esta função utiliza os primeiros 8 bytes, sendo que os primeiros τ (peso de c) bits vão corresponder ao *sign bits*. Cada byte do output de *genHash256* vai corresponder a um inteiro em formato *little endian* que será gerado pela função `genIntegerJ`. Sendo j os valores inteiros gerados e s os *signs bits*, de seguida será aplicado o algoritmo *sample in ball* tal como se encontra na página 10 da documentação.

No final, a variavél z irá corresponder à expressão $z = y + c * \mathbf{s}_1$ e a variável r_0 irá ser obtida através da função `LowBits` que tem como objetivo extrair os *lower-order bits* do output da função *Decompose*. A variavél h será obtida através da função `MakeHint` que tem como objetivo calcular um *hint* que será usado para recuperar high-order bits da soma $\mathbf{w} - c\mathbf{s}_2 + c\mathbf{t}_0$. Caso a condição apresentada em cima não se verificar, então o ciclo começa novamente com $k = k + l$. Caso a condição se verificar, então a assinatura é dada por $\sigma = (z, h, \tilde{c})$.

Verificação da assinatura: A função `Verify` tem como objetivo verificar a autenticidade da assinatura σ recebida como parâmetro quando associada à mensagem M utilizando para isso a a

chave pública pk. Para isso começamos por determinar a matriz A através da função *ExpandA* e da variavél ρ. De seguida calculamos a variável $y = CRH(CRH(\rho \mid\mid \mathbf{t}_1) \mid\mid M)$. Para calcularmos a variável c utilizamos a função *SampleInBall* e a variável $\tilde{c}$. Estas variáveis todas serão utilizadas na função `UseHint` para assim determinar $\mathbf{w}_1' := \text{UseHint}(\mathbf{h}, \mathbf{Az} - c\mathbf{t}_1 \cdot 2^d, 2\gamma_2)$. Esta função tem como objetivo usar o *hint* para recuperar *high-order bits* da soma apresentada. Para verificar a autenticidade da assinatura verificamos a condição $||\mathbf{z}||_\infty < \gamma_1 - \beta$ **and** $\tilde{c} = \text{H}(\mu \mid\mid \mathbf{w}_1')$ **and** # of 1's in $\mathbf{h} \leq \omega$

Notas: As funções *Decompose*, *UseHint* e *MakeHint* foram implementas de forma a devolverem e receberem vetores para assim ir de encontro com os tipos de dados apresentados nos restantes passos do algoritmo. É também necessário ter emk atenção que a função *Power2Round*, entre outras, recebe como parâmetro vetores e que para isso utiliza as funções `modVector` e `modpmVector` que aplica a operação de *mod* e $mod^{\pm}$, respetivamente, para cada elemento do vetor de polinómios. A função `modpm` tem como objetivo implementar a operação $mod^{\pm}$, segundo se encontra descrito na página 9 da documentação.

[11]:
```
class DILITHIUM:

    def __init__(self):
        self.q,self.n,self.d,self.c,self.y1,self.y2,self.k,self.l,self.nn,self.
↪w,self.b,self.R,self.Rq = self.setup()

    #Parâmetros da técnica DILITHIUM - NIST level 5 - 5+
    def setup(self):

        q = 8380417
        n = 256
        d = 13
        c = 60
        y1 = 2**19
        y2 = (q - 1)//32
        k = 9
        l = 8
        nn = 2
        b = 120
        w = 85

        Z.<x> = ZZ[]
        R.<x> = QuotientRing(Z,Z.ideal(x^n+1))

        Zq.<x> = GF(q)[]
        fi     = x^n + 1
        Rq.<x> = QuotientRing(Zq,Zq.ideal(fi))

        return q,n,d,c,y1,y2,k,l,nn,w,b,R,Rq
```

```
    # ------------------------------------ FUNÇÕES AUXLIARES␣
↪------------------------------------ #

    #Implementação do mod+- segundo "Modular Reductions" - pag 9
    def modpm(self,r,a):

        if a%2 == 0:
            limit = a/2
        else:
            limit = (a-1)/2

        rMod = r % a

        if rMod > limit:
            rMod -= a

        return rMod

    # Implementação do mod para todos os elementos de um vetor
    def modVector(self, v,a):

        rAux = []

        for i in v:
            iAux = []
            for j in i:
                iAux.append(mod(j,a))

            rAux.append(self.R(iAux))

        return vector(rAux)

    # Implementação do mod+- para todos os elementos de um vetor
    def modpmVector(self, v,a):

        rAux = []

        for i in v:
            iAux = []
            for j in i:
                iAux.append(self.modpm(j,a))

            rAux.append(self.R(iAux))

        return vector(rAux)

    #Implementação de ||v||∞ segundo "Sizes of elements" - pag 9
```

```
    def normInf(self, v):

        maxsV = []

        # ||v||∞ = max(||vi||∞)
        for vi in v:
            maxsVi = []

            #||vi||∞ = max(||i||∞)
            for i in vi:
                #||i||∞ = |i mod ± q|
                maxsVi.append(self.modpm(i,self.q))

            maxsV.append(max(maxsVi))

        return max(maxsV)

    #Geração das samples s1 e s2 - pag 9
    def vectorGen(self,c):

        # We will write S  to denote all elements w   R such that ||w||∞    .

        hashS = self.H(c, self.l*self.n + self.k*self.n)
        index = 0
        s1 = []

        for i in range(self.l):
            poli = []

            for j in range(self.n):
                num = hashS[index]
                index += 1

                poli.append(self.modpm(num,self.nn))

            s1.append(self.R(poli))

        s2 = []

        for i in range(self.k):
            poli = []

            for j in range(self.n):
                num = hashS[index]
                index += 1

                poli.append(self.modpm(num,self.nn))
```

```
            s2.append(self.R(poli))

        return (vector(s1),vector(s2))

    #Implementação da Função H para geração do triplo ( ,  , K)
    def tripleH(self, C):

        #256*3 bits = 96 bytes
        digest = hashes.Hash(hashes.SHAKE256(int(96)))
        digest.update(C)
        buffer = digest.finalize()

        return (buffer[:32], buffer[32:64], buffer[64:])

    #Função auxiliar que permite transformar a matrix A de Rq para R
    def fromRqToR(self, A):

        mtx = []

        for row in A:

            newRow = []

            for elem in row:

                newRow.append(self.R(elem))

            mtx.append(newRow)

        return Matrix(mtx)

    # ------------------------------------ ALGORITMOS DE SUPORTE (PAG 12) ␣
↪------------------------------------ #

    #Função que parte elementos de Rq Bit-wise - pag 11-12
    def Power2Round(self,r,d):

        r = self.modVector(r,self.q)
        r0 = self.modpmVector(r,2^d)

        return ((r - r0)/2^d, r0)

    #Função que reduz o tamanho da chave publica através da geração dos␣
↪"higher-order" and "lower-order" bits
    # dos elementos pertencentes a Zq - Aplicada a vetores
    def Decompose(self,r,alpha):
```

```
        r = self.modVector(r,self.q)
        r0 = self.modpmVector(r,alpha)

        rElems = []
        r0Elems = []

        for ri, r0i in zip(r, r0):

            r1Coefs = []
            r0Coefs = []

            for rij, r0ij in zip(ri, r0i):

                if (rij - r0ij == self.q - 1):
                    r1ij = 0
                    r0ij = r0ij-1
                else:
                    r1ij = (rij - r0ij)/alpha

                r1Coefs.append(r1ij)
                r0Coefs.append(r0ij)

            rElems.append(self.R(r1Coefs))
            r0Elems.append(self.R(r0Coefs))

        return (vector(rElems),vector(r0Elems))

    #Função que extrai os "higher-order" bits do output de Decompose
    def HighBits(self,r,alpha):

        (r1,r0) = self.Decompose(r,alpha)

        return r1

    #Função que extrai os "lower-order" bits do output de Decompose
    def LowBits(self,r,alpha):

        (r1,r0) = self.Decompose(r,alpha)

        return r0

    #Função que calcula um hint que será usado para recuperar high-order bits
↪da soma - pag 11 e 12
    # Aplicada para vetores
    def MakeHint(self,z,r,alpha):
```

```
        r1 = self.HighBits(r,alpha)
        v1 = self.HighBits(r + z,alpha)

        vec = []
        for r1i, v1i in zip(r1, v1):
            for r1ij, v1ij in zip(r1i, v1i):
                vec.append(r1ij != v1ij)

        return vec

    #Função que usa o hint para recuperar high-order bits da soma - pag 11 e 12
    # Aplicada para vetores
    def UseHint(self,h,r,alpha):

        m = (self.q - 1)/alpha
        (r1,r0) = self.Decompose(r,alpha)

        rAux = []
        i = 0

        for r0i, r1i in zip(r0,r1):
            elem = []

            for r0ij, r1ij in zip(r0i,r1i):
                if h[i] :
                    if r0ij > 0:
                        elem.append(mod((r1ij + 1),m))

                    else:
                         elem.append(mod((r1ij - 1),m))
                else:
                    elem.append(r1ij)

                i += 1

            rAux.append(self.R(elem))

        return vector(rAux)

    # ------------------------------------------------ FUNÇÕES HASH ␣
↪------------------------------------------- #

    #Implementação da função H - pag 12
    def H(self, m, length):

        digest = hashes.Hash(hashes.SHAKE256(int(length)))
        digest.update(m)
```

```
        h = digest.finalize()

        return h

    #Implementação da "Collision resistant hash" - pag 20
    def CRH(self,seed):

        prf = hashes.Hash(hashes.SHAKE256(int(48)))
        prf.update(seed)

        return prf.finalize()

    # ------------------------------------------ FUNÇÕES DE SAMPLING ␣
↪----------------------------------------- #

    #Implementação da função "Expanding the Matrix A" - pag 19
    def ExpandA(self,ro):

        #Função Hash a ser utilizada neste contexto
        def genHash128(seed):

            digest = hashes.Hash(hashes.SHAKE128(int(self.n-1)))
            digest.update(seed)

            return digest.finalize()

        #Gera o integer segundo as regras apresentadas
        def genInteger(offset, seedShake):

            if(len(offset)<3):
                offset = seedShake

            threeBytes = bytearray(offset[:3])
            threeBytes[2]  &= 0x7f
            threeBytes = bytes(threeBytes)

            number = int.from_bytes(threeBytes, "little")

            return number, offset[3:]

        A = []

        for i in range(self.k):
            row = []
            for j in range(self.l):
                coefs = []
                intTwoBytes = int(self.n*i + j).to_bytes(2, "little")
```

```
                hashShake = genHash128(ro + intTwoBytes)
                seedShake = hashShake

                for cof in range(self.n): #rejection sampling??
                    number, hashShake = genInteger(hashShake, seedShake)

                    coefs.append(number)
                row.append(self.Rq(coefs))
            A.append(row)

        return Matrix(A)

    #Implementação da função "Hashing to a Ball" - pag 19
    def SampleInBall(self,ro):

        #Função Hash a ser utilizada neste contexto
        def genHash256(seed):

            digest = hashes.Hash(hashes.SHAKE256(int(self.n)))
            digest.update(seed)

            return digest.finalize()

        #Função que retira dos primeiros 8 bytes os sign bits
        def genSignBits(seed):

            digest = int.from_bytes(seed[:8], 'little')

            bits = [int(digit) for digit in list(ZZ(digest).binary())]

            return bits[:self.c], seed[8:]

        #Função que gera o valor de j
        def genIntegerJ(offset, seedShake):

            if(len(offset)<1):
                offset = seedShake

            number = int.from_bytes(bytes(offset[0]), "little")

            return number, offset[1:]

        c = [0]*self.n
        hashShake = genHash256(ro)
        seedShake = hashShake

        signBits, hashShake = genSignBits(hashShake)
```

```
    for i in range(self.n - self.c, self.n): #rejection sampling??

        j, hashShake = genIntegerJ(hashShake,seedShake)
        s = signBits[i-self.n + self.c]
        c[i] = c[j]
        c[j] = (-1)^s

    return self.R(c)

#Implementação da função "Sampling the vectors" - pag 20
def ExpandMask(self, ro, k):

    #Função Hash a ser utilizada neste contexto
    def genHash256(seed):

        digest = hashes.Hash(hashes.SHAKE256(int(self.n-1)))
        digest.update(seed)

        return digest.finalize()

    #Função que gera o inteiro segundo regras apresentadas
    def genIntegerMask(offset, seedShake):

        if(len(offset)<3):
            offset = seedShake

        threeBytes = bytearray(offset[:3])

        if self.y1 == 2**17:
            threeBytes[2]  &= 0x3
        else:
            threeBytes[2]  &= 0xf

        threeBytes = bytes(threeBytes)

        number = int.from_bytes(threeBytes, "little")

        return number, offset[3:]

    y = []

    for i in range(self.l):
        coefs = []
        intTwoBytes = int(k + i).to_bytes(2, "little")
        hashShake = genHash256(ro + intTwoBytes)
        seedShake = hashShake
```

```
            for cof in range(self.n):
                number, hashShake = genIntegerMask(hashShake, seedShake)

                coefs.append(number - (self.y1 -1))
            y.append(self.R(coefs))

        return vector(y)

    # ---------------------------------------- FUNÇÕES PRINCIPAIS (PAG 13) ␣
↪------------------------------------ #

    #FUNÇÃO: Gera o par de chaves (publica e privada) para a assinatura e␣
↪verificação
    def Gen(self):

        #   ← {0, 1}^256
        C = os.urandom(32)

        #( ,  , K )   {0, 1}^256×3 := H( )
        (ro, c, K) = self.tripleH(C)

        #(s1,s2)   S ^l × S ^k := H( )
        (s1,s2) = self.vectorGen(c)

        #A   Rq^k×l := ExpandA( )
        A = self.ExpandA(ro)

        #t := A*s1 + s2
        t = A * s1 + s2

        #(t1,t0) := Power2Round(t, d)
        (t1, t0) = self.Power2Round(t,self.d)

        #tr   {0, 1}^384 := CRH( || t1)
        tr = self.CRH(ro + dumps(t1))

        return (ro, t1),(ro, K, tr, s1, s2, t0) #pk, sk

    #FUNÇÃO: Assina a mensagem a enviar M utilizando a chave privada sk
    def Sign(self, sk, M):

        (ro, K, tr, s1, s2, t0) = sk

        #A   Rq^k×l := ExpandA( )
        A = self.ExpandA(ro)
```

```
        #   {0, 1}^384:= CRH(tr || M)
        yy = self.CRH(tr + M)

        #  := 0
        k = 0

        #(z, h) :=
        (z,h) = (None, None)

        # '   {0,1}^384 := CRH(K ||  )
        roNew = self.CRH(K + yy)

        while z == None or h == None:

            #y   S ^l := ExpandMask( ',  )
            y = self.ExpandMask(roNew, k)

            #w := Ay
            w = A * y

            #w1 := HighBits(w, 2* 2 )
            w1 = self.HighBits(w, 2 * self.y2)

            #c'   {0,1}^256 := H(  || w1)
            cAux = self.H(yy + dumps(w1), 32)
            #c   B  := SampleInBall(c')
            c = self.SampleInBall(cAux)

            #z := y + cs 1
            z = y + c * s1

            #r0 := LowBits(w - c*s2 , 2* 2)
            r0 = self.LowBits(w - c * s2, 2 * self.y2)


            #||z||∞   1 -   or ||r0||∞   2 -
            if self.normInf(z) >= (self.y1 - self.b) and self.normInf(r0) >=␣
↪(self.y2 - self.b):

                #(z, h) :=
                (z,h) = (None, None)

            else:

                #h := MakeHint(-c*t0 , w - c*s2 + c*t0 , 2* 2)
                h = self.MakeHint( -c * t0, w - c * s2 + c * t0, 2 * self.y2)
```

```
                #||c*t0||∞   2 or the # of 1's in h is greater than
                if self.normInf(c * t0) >= self.y2 or h.count(True) > self.w:

                    #(z, h) :=
                    (z,h) = (None, None)

            #  :=   + l
            k = k + self.l

        return (z,h,cAux) #sigma

    #FUNÇÃO: Verifica a assinatura na mensagem utilizando a chave publica pk
    def Verify(self,pk,M,sigma):

        (ro, t1) = pk
        (z,h,cAux) = sigma

        #A   Rq^k×l := ExpandA( )
        A = self.ExpandA(ro)

        #   {0, 1}^384 := CRH(CRH(  + t1) + M )
        y = self.CRH(self.CRH(ro + dumps(t1)) + M)

        #c := SampleInBall(c')
        c = self.SampleInBall(cAux)

        #w' = UseHint(h, A*z - c*t1 * 2^d , 2*2)
        w1 = self.UseHint(h, (self.fromRqToR(A) * z) - (c * t1 * 2^self.d), 2 *␣
↪self.y2)

        #||z||∞ <  1 -   and c' = H(  || w') and J# of 1's in h is
        return self.normInf(z) < (self.y1-self.b) and cAux == self.H(y +␣
↪dumps(w1), 32) and h.count(True) <= self.w
```

CENÁRIO DE TESTE

[12]:
```
dlth = DILITHIUM()

message = "Mensagem a ser assinada"
messageErrada = "Mensagem a ser assinado"

public, private = dlth.Gen()

signature = dlth.Sign(private, message.encode())

print("SIGNATURE: ")
print(signature)
```

```
result = dlth.Verify(public, message.encode(), signature)

print("MENSAGEM VÁLIDA")
print(message)

if(result):
    print("Assinatura Válida!")
else:
    print("Assinatura Inválida!")

result2 = dlth.Verify(public, messageErrada.encode(), signature)

print("MENSAGEM INVÁLIDA")
print(messageErrada)

if(result2):
    print("Assinatura Válida!")
else:
    print("Assinatura Inválida!")
```

```
SIGNATURE:
((-249504*x^255 - 475517*x^254 + 217354*x^253 - 234626*x^252 - 110688*x^251 -
48290*x^250 - 165675*x^249 + 319970*x^248 + 259210*x^247 - 243678*x^246 +
349165*x^245 - 25471*x^244 + 152009*x^243 + 91610*x^242 + 1985*x^241 +
184334*x^240 - 141553*x^239 - 148266*x^238 - 148670*x^237 - 215122*x^236 -
190700*x^235 - 3391*x^234 + 165014*x^233 + 522999*x^232 + 471830*x^231 -
343405*x^230 + 25150*x^229 + 206208*x^228 + 107982*x^227 + 521126*x^226 +
508948*x^225 + 176728*x^224 + 261323*x^223 + 343830*x^222 + 462364*x^221 +
306977*x^220 + 442765*x^219 + 162741*x^218 - 514059*x^217 + 272503*x^216 +
168724*x^215 - 6404*x^214 + 331803*x^213 + 451487*x^212 - 467475*x^211 +
114059*x^210 + 23940*x^209 + 189118*x^208 + 230715*x^207 - 284342*x^206 -
258016*x^205 - 149396*x^204 - 460883*x^203 + 498964*x^202 + 43637*x^201 -
455107*x^200 + 387854*x^199 + 445249*x^198 + 300294*x^197 - 419104*x^196 -
283384*x^195 - 159769*x^194 + 214683*x^193 - 398496*x^192 + 252773*x^191 +
18013*x^190 - 222009*x^189 + 253410*x^188 - 205964*x^187 + 354210*x^186 -
215468*x^185 + 512951*x^184 - 228447*x^183 + 163617*x^182 + 280168*x^181 -
41626*x^180 - 420812*x^179 - 481389*x^178 - 137637*x^177 + 157713*x^176 +
521782*x^175 + 359735*x^174 + 496289*x^173 - 167282*x^172 + 201521*x^171 -
249504*x^170 - 475518*x^169 + 217349*x^168 - 234626*x^167 - 110686*x^166 -
48293*x^165 - 165665*x^164 + 319968*x^163 + 259219*x^162 - 243667*x^161 +
349163*x^160 - 25468*x^159 + 152011*x^158 + 91611*x^157 + 1988*x^156 +
184336*x^155 - 141553*x^154 - 148261*x^153 - 148653*x^152 - 215122*x^151 -
190694*x^150 - 3388*x^149 + 165008*x^148 + 522990*x^147 + 471834*x^146 -
343400*x^145 + 25147*x^144 + 206213*x^143 + 107990*x^142 + 521132*x^141 +
508949*x^140 + 176739*x^139 + 261325*x^138 + 343841*x^137 + 462373*x^136 +
306975*x^135 + 442758*x^134 + 162746*x^133 - 514065*x^132 + 272499*x^131 +
```

```
168732*x^130 - 6383*x^129 + 331806*x^128 + 451497*x^127 - 467474*x^126 +
114064*x^125 + 23942*x^124 + 189109*x^123 + 230710*x^122 - 284336*x^121 -
258011*x^120 - 149394*x^119 - 460881*x^118 + 498964*x^117 + 43640*x^116 -
455108*x^115 + 387853*x^114 + 445250*x^113 + 300297*x^112 - 419100*x^111 -
283388*x^110 - 159774*x^109 + 214683*x^108 - 398498*x^107 + 252781*x^106 +
18017*x^105 - 222005*x^104 + 253412*x^103 - 205961*x^102 + 354200*x^101 -
215473*x^100 + 512943*x^99 - 228441*x^98 + 163611*x^97 + 280172*x^96 -
41616*x^95 - 420808*x^94 - 481388*x^93 - 137635*x^92 + 157706*x^91 + 521779*x^90
+ 359740*x^89 + 496286*x^88 - 167287*x^87 + 201528*x^86 - 249507*x^85 -
475516*x^84 + 217354*x^83 - 234620*x^82 - 110686*x^81 - 48293*x^80 - 165671*x^79
+ 319972*x^78 + 259206*x^77 - 243685*x^76 + 349163*x^75 - 25470*x^74 +
152017*x^73 + 91610*x^72 + 1985*x^71 + 184334*x^70 - 141554*x^69 - 148264*x^68 -
148664*x^67 - 215123*x^66 - 190691*x^65 - 3385*x^64 + 165017*x^63 + 522998*x^62
+ 471839*x^61 - 343404*x^60 + 25140*x^59 + 206211*x^58 + 107984*x^57 +
521138*x^56 + 508950*x^55 + 176733*x^54 + 261325*x^53 + 343850*x^52 +
462369*x^51 + 306981*x^50 + 442763*x^49 + 162746*x^48 - 514052*x^47 +
272494*x^46 + 168722*x^45 - 6400*x^44 + 331803*x^43 + 451484*x^42 - 467478*x^41
+ 114063*x^40 + 23945*x^39 + 189119*x^38 + 230711*x^37 - 284332*x^36 -
258014*x^35 - 149393*x^34 - 460889*x^33 + 498960*x^32 + 43637*x^31 - 455103*x^30
+ 387855*x^29 + 445242*x^28 + 300298*x^27 - 419099*x^26 - 283387*x^25 -
159767*x^24 + 214683*x^23 - 398501*x^22 + 252781*x^21 + 18008*x^20 - 222012*x^19
+ 253407*x^18 - 205965*x^17 + 354202*x^16 - 215469*x^15 + 512955*x^14 -
228438*x^13 + 163607*x^12 + 280167*x^11 - 41625*x^10 - 420813*x^9 - 481382*x^8 -
137633*x^7 + 157713*x^6 + 521783*x^5 + 359744*x^4 + 496288*x^3 - 167287*x^2 +
201529*x - 249509, 315960*x^255 - 104376*x^254 - 402413*x^253 + 171757*x^252 +
196842*x^251 - 121148*x^250 + 34118*x^249 - 319473*x^248 - 58178*x^247 -
513207*x^246 - 200015*x^245 - 361404*x^244 - 137064*x^243 + 394840*x^242 -
49526*x^241 + 208417*x^240 - 301809*x^239 + 190859*x^238 + 289268*x^237 +
481817*x^236 - 202803*x^235 - 28627*x^234 + 126712*x^233 - 403369*x^232 -
363012*x^231 + 254622*x^230 - 300303*x^229 + 184451*x^228 + 394798*x^227 -
113536*x^226 + 256808*x^225 + 465370*x^224 + 482153*x^223 + 350363*x^222 -
29747*x^221 - 73256*x^220 + 319224*x^219 + 173079*x^218 + 359238*x^217 -
445046*x^216 - 117030*x^215 - 88205*x^214 - 367048*x^213 - 350924*x^212 +
163171*x^211 - 148797*x^210 - 247448*x^209 + 177372*x^208 - 20808*x^207 -
458470*x^206 + 41229*x^205 - 167512*x^204 + 213641*x^203 - 2646*x^202 -
391719*x^201 + 115381*x^200 - 58119*x^199 + 290335*x^198 + 341832*x^197 +
10984*x^196 - 311205*x^195 + 499052*x^194 - 483708*x^193 - 277978*x^192 -
155476*x^191 - 218431*x^190 - 116739*x^189 + 152126*x^188 - 289671*x^187 +
456058*x^186 - 249345*x^185 + 443026*x^184 - 244909*x^183 + 135196*x^182 +
56816*x^181 - 118607*x^180 - 220199*x^179 + 224813*x^178 + 370781*x^177 -
331006*x^176 + 321481*x^175 + 357323*x^174 - 366928*x^173 + 52899*x^172 -
237487*x^171 + 315969*x^170 - 104371*x^169 - 402422*x^168 + 171757*x^167 +
196837*x^166 - 121149*x^165 + 34117*x^164 - 319470*x^163 - 58175*x^162 -
513195*x^161 - 200017*x^160 - 361405*x^159 - 137060*x^158 + 394844*x^157 -
49529*x^156 + 208416*x^155 - 301810*x^154 + 190870*x^153 + 289269*x^152 +
481813*x^151 - 202807*x^150 - 28630*x^149 + 126714*x^148 - 403369*x^147 -
363016*x^146 + 254626*x^145 - 300307*x^144 + 184452*x^143 + 394793*x^142 -
113531*x^141 + 256816*x^140 + 465379*x^139 + 482161*x^138 + 350370*x^137 -
```

```
29737*x^136 - 73247*x^135 + 319220*x^134 + 173076*x^133 + 359238*x^132 -
445042*x^131 - 117027*x^130 - 88205*x^129 - 367046*x^128 - 350920*x^127 +
163176*x^126 - 148793*x^125 - 247453*x^124 + 177372*x^123 - 20803*x^122 -
458471*x^121 + 41230*x^120 - 167505*x^119 + 213643*x^118 - 2645*x^117 -
391715*x^116 + 115375*x^115 - 58115*x^114 + 290332*x^113 + 341824*x^112 +
10988*x^111 - 311206*x^110 + 499047*x^109 - 483700*x^108 - 277977*x^107 -
155478*x^106 - 218421*x^105 - 116735*x^104 + 152132*x^103 - 289666*x^102 +
456054*x^101 - 249352*x^100 + 443027*x^99 - 244911*x^98 + 135196*x^97 +
56819*x^96 - 118610*x^95 - 220200*x^94 + 224808*x^93 + 370781*x^92 - 331009*x^91
+ 321475*x^90 + 357330*x^89 - 366927*x^88 + 52897*x^87 - 237481*x^86 +
315964*x^85 - 104383*x^84 - 402415*x^83 + 171758*x^82 + 196828*x^81 -
121147*x^80 + 34122*x^79 - 319477*x^78 - 58172*x^77 - 513201*x^76 - 200018*x^75
- 361392*x^74 - 137057*x^73 + 394845*x^72 - 49525*x^71 + 208414*x^70 -
301814*x^69 + 190859*x^68 + 289268*x^67 + 481810*x^66 - 202802*x^65 - 28632*x^64
+ 126709*x^63 - 403366*x^62 - 363010*x^61 + 254620*x^60 - 300310*x^59 +
184453*x^58 + 394804*x^57 - 113529*x^56 + 256812*x^55 + 465379*x^54 +
482160*x^53 + 350376*x^52 - 29742*x^51 - 73257*x^50 + 319229*x^49 + 173080*x^48
+ 359238*x^47 - 445047*x^46 - 117028*x^45 - 88206*x^44 - 367039*x^43 -
350922*x^42 + 163175*x^41 - 148790*x^40 - 247452*x^39 + 177373*x^38 - 20810*x^37
- 458478*x^36 + 41231*x^35 - 167507*x^34 + 213634*x^33 - 2651*x^32 - 391715*x^31
+ 115379*x^30 - 58114*x^29 + 290330*x^28 + 341827*x^27 + 10985*x^26 -
311193*x^25 + 499049*x^24 - 483699*x^23 - 277975*x^22 - 155475*x^21 -
218433*x^20 - 116742*x^19 + 152131*x^18 - 289666*x^17 + 456050*x^16 -
249355*x^15 + 443022*x^14 - 244910*x^13 + 135201*x^12 + 56816*x^11 - 118610*x^10
- 220199*x^9 + 224814*x^8 + 370781*x^7 - 331012*x^6 + 321471*x^5 + 357324*x^4 -
366926*x^3 + 52906*x^2 - 237482*x + 315966, 436*x^255 + 335906*x^254 +
444640*x^253 + 440570*x^252 - 96007*x^251 + 278778*x^250 - 277762*x^249 +
13705*x^248 - 60293*x^247 - 468929*x^246 + 46222*x^245 + 497482*x^244 +
13503*x^243 - 58508*x^242 - 317058*x^241 + 485429*x^240 + 259814*x^239 -
237122*x^238 + 190678*x^237 - 375218*x^236 - 128885*x^235 - 462046*x^234 +
439371*x^233 - 175834*x^232 - 180121*x^231 - 74368*x^230 + 388223*x^229 -
151194*x^228 - 181855*x^227 + 258309*x^226 - 3919*x^225 + 29145*x^224 -
306121*x^223 - 1176*x^222 + 253640*x^221 - 266120*x^220 + 486304*x^219 -
137367*x^218 - 445655*x^217 + 495170*x^216 + 123116*x^215 - 8823*x^214 -
500754*x^213 - 370324*x^212 + 125820*x^211 + 113847*x^210 - 374374*x^209 +
228856*x^208 + 83657*x^207 + 416127*x^206 - 404957*x^205 + 98247*x^204 +
216305*x^203 + 81756*x^202 - 459720*x^201 - 198609*x^200 - 327055*x^199 +
272381*x^198 - 42004*x^197 - 236573*x^196 - 140536*x^195 + 398475*x^194 +
486064*x^193 + 456729*x^192 - 459628*x^191 - 325323*x^190 - 249574*x^189 +
442847*x^188 - 485125*x^187 + 267175*x^186 - 246960*x^185 - 91355*x^184 -
397070*x^183 - 413313*x^182 + 139434*x^181 + 154600*x^180 + 2271*x^179 -
206974*x^178 + 453717*x^177 - 465975*x^176 - 474768*x^175 + 191753*x^174 +
438217*x^173 + 29142*x^172 + 248504*x^171 + 437*x^170 + 335915*x^169 +
444634*x^168 + 440564*x^167 - 96016*x^166 + 278777*x^165 - 277754*x^164 +
13700*x^163 - 60288*x^162 - 468909*x^161 + 46225*x^160 + 497479*x^159 +
13507*x^158 - 58509*x^157 - 317050*x^156 + 485423*x^155 + 259809*x^154 -
237114*x^153 + 190684*x^152 - 375225*x^151 - 128887*x^150 - 462048*x^149 +
439378*x^148 - 175830*x^147 - 180119*x^146 - 74370*x^145 + 388220*x^144 -
```

```
151195*x^143 - 181857*x^142 + 258297*x^141 - 3914*x^140 + 29162*x^139 -
306118*x^138 - 1163*x^137 + 253651*x^136 - 266112*x^135 + 486314*x^134 -
137367*x^133 - 445646*x^132 + 495174*x^131 + 123116*x^130 - 8820*x^129 -
500760*x^128 - 370338*x^127 + 125832*x^126 + 113855*x^125 - 374375*x^124 +
228860*x^123 + 83663*x^122 + 416128*x^121 - 404965*x^120 + 98233*x^119 +
216295*x^118 + 81765*x^117 - 459716*x^116 - 198611*x^115 - 327058*x^114 +
272392*x^113 - 42000*x^112 - 236585*x^111 - 140543*x^110 + 398476*x^109 +
486058*x^108 + 456729*x^107 - 459626*x^106 - 325332*x^105 - 249567*x^104 +
442854*x^103 - 485133*x^102 + 267171*x^101 - 246954*x^100 - 91359*x^99 -
397076*x^98 - 413324*x^97 + 139431*x^96 + 154602*x^95 + 2266*x^94 - 206978*x^93
+ 453721*x^92 - 465971*x^91 - 474767*x^90 + 191758*x^89 + 438214*x^88 +
29144*x^87 + 248505*x^86 + 426*x^85 + 335910*x^84 + 444638*x^83 + 440569*x^82 -
96010*x^81 + 278776*x^80 - 277753*x^79 + 13710*x^78 - 60295*x^77 - 468912*x^76 +
46220*x^75 + 497480*x^74 + 13511*x^73 - 58512*x^72 - 317055*x^71 + 485428*x^70 +
259812*x^69 - 237116*x^68 + 190682*x^67 - 375221*x^66 - 128883*x^65 -
462049*x^64 + 439371*x^63 - 175824*x^62 - 180114*x^61 - 74365*x^60 + 388223*x^59
- 151197*x^58 - 181853*x^57 + 258309*x^56 - 3924*x^55 + 29154*x^54 - 306121*x^53
- 1168*x^52 + 253653*x^51 - 266117*x^50 + 486313*x^49 - 137363*x^48 -
445652*x^47 + 495168*x^46 + 123123*x^45 - 8818*x^44 - 500756*x^43 - 370331*x^42
+ 125835*x^41 + 113856*x^40 - 374369*x^39 + 228858*x^38 + 83666*x^37 +
416130*x^36 - 404957*x^35 + 98240*x^34 + 216294*x^33 + 81760*x^32 - 459719*x^31
- 198620*x^30 - 327053*x^29 + 272392*x^28 - 41999*x^27 - 236586*x^26 -
140542*x^25 + 398472*x^24 + 486057*x^23 + 456726*x^22 - 459633*x^21 -
325330*x^20 - 249564*x^19 + 442854*x^18 - 485125*x^17 + 267174*x^16 -
246947*x^15 - 91349*x^14 - 397082*x^13 - 413312*x^12 + 139429*x^11 + 154591*x^10
+ 2274*x^9 - 206979*x^8 + 453710*x^7 - 465964*x^6 - 474763*x^5 + 191751*x^4 +
438216*x^3 + 29139*x^2 + 248501*x + 429, -328588*x^255 - 83379*x^254 -
353564*x^253 + 183147*x^252 + 74151*x^251 + 17717*x^250 + 195258*x^249 -
438933*x^248 - 475335*x^247 + 222716*x^246 + 268351*x^245 - 439614*x^244 +
289161*x^243 - 29693*x^242 - 248342*x^241 - 259505*x^240 - 336404*x^239 +
240860*x^238 - 485746*x^237 + 478122*x^236 - 406327*x^235 - 331901*x^234 -
286204*x^233 - 94331*x^232 + 465746*x^231 + 271991*x^230 + 400027*x^229 +
305826*x^228 + 460515*x^227 - 63715*x^226 - 341157*x^225 + 129833*x^224 -
39235*x^223 - 57495*x^222 - 470893*x^221 + 326041*x^220 + 272317*x^219 +
271609*x^218 - 69706*x^217 - 147346*x^216 + 36731*x^215 - 132228*x^214 -
511283*x^213 - 368507*x^212 + 62728*x^211 + 69000*x^210 + 259153*x^209 -
427972*x^208 - 351552*x^207 - 19430*x^206 - 387050*x^205 - 94801*x^204 +
155109*x^203 - 345476*x^202 - 379791*x^201 + 450997*x^200 - 95795*x^199 -
110620*x^198 - 203923*x^197 - 520866*x^196 + 64865*x^195 + 56953*x^194 +
345111*x^193 + 339695*x^192 + 331489*x^191 - 60436*x^190 - 479919*x^189 +
217047*x^188 - 394408*x^187 + 144095*x^186 - 346859*x^185 + 458846*x^184 +
91606*x^183 + 429361*x^182 - 277562*x^181 - 236359*x^180 - 260555*x^179 -
219172*x^178 - 229071*x^177 - 135898*x^176 - 379946*x^175 - 431261*x^174 -
24831*x^173 + 389942*x^172 + 506511*x^171 - 328594*x^170 - 83388*x^169 -
353576*x^168 + 183151*x^167 + 74162*x^166 + 17709*x^165 + 195266*x^164 -
438924*x^163 - 475334*x^162 + 222715*x^161 + 268358*x^160 - 439618*x^159 +
289174*x^158 - 29688*x^157 - 248335*x^156 - 259496*x^155 - 336396*x^154 +
240863*x^153 - 485747*x^152 + 478127*x^151 - 406326*x^150 - 331898*x^149 -
```

```
286205*x^148 - 94327*x^147 + 465746*x^146 + 271999*x^145 + 400029*x^144 +
305824*x^143 + 460526*x^142 - 63712*x^141 - 341158*x^140 + 129838*x^139 -
39230*x^138 - 57493*x^137 - 470896*x^136 + 326038*x^135 + 272325*x^134 +
271621*x^133 - 69698*x^132 - 147334*x^131 + 36742*x^130 - 132223*x^129 -
511269*x^128 - 368504*x^127 + 62719*x^126 + 69002*x^125 + 259154*x^124 -
427982*x^123 - 351546*x^122 - 19432*x^121 - 387056*x^120 - 94797*x^119 +
155111*x^118 - 345479*x^117 - 379792*x^116 + 450990*x^115 - 95803*x^114 -
110623*x^113 - 203919*x^112 - 520865*x^111 + 64864*x^110 + 56951*x^109 +
345111*x^108 + 339694*x^107 + 331488*x^106 - 60432*x^105 - 479928*x^104 +
217042*x^103 - 394402*x^102 + 144086*x^101 - 346856*x^100 + 458846*x^99 +
91606*x^98 + 429367*x^97 - 277553*x^96 - 236362*x^95 - 260553*x^94 - 219172*x^93
- 229075*x^92 - 135897*x^91 - 379942*x^90 - 431263*x^89 - 24832*x^88 +
389944*x^87 + 506507*x^86 - 328582*x^85 - 83378*x^84 - 353572*x^83 + 183156*x^82
+ 74155*x^81 + 17708*x^80 + 195254*x^79 - 438929*x^78 - 475344*x^77 +
222712*x^76 + 268358*x^75 - 439613*x^74 + 289172*x^73 - 29688*x^72 - 248334*x^71
- 259498*x^70 - 336395*x^69 + 240869*x^68 - 485741*x^67 + 478122*x^66 -
406326*x^65 - 331905*x^64 - 286203*x^63 - 94330*x^62 + 465746*x^61 + 271998*x^60
+ 400027*x^59 + 305825*x^58 + 460525*x^57 - 63718*x^56 - 341158*x^55 +
129831*x^54 - 39236*x^53 - 57491*x^52 - 470894*x^51 + 326042*x^50 + 272321*x^49
+ 271619*x^48 - 69692*x^47 - 147342*x^46 + 36737*x^45 - 132225*x^44 -
511274*x^43 - 368506*x^42 + 62728*x^41 + 69000*x^40 + 259157*x^39 - 427975*x^38
- 351555*x^37 - 19436*x^36 - 387048*x^35 - 94807*x^34 + 155102*x^33 -
345487*x^32 - 379794*x^31 + 450995*x^30 - 95798*x^29 - 110626*x^28 - 203919*x^27
- 520859*x^26 + 64864*x^25 + 56957*x^24 + 345105*x^23 + 339702*x^22 +
331488*x^21 - 60432*x^20 - 479926*x^19 + 217048*x^18 - 394398*x^17 + 144089*x^16
- 346861*x^15 + 458840*x^14 + 91613*x^13 + 429365*x^12 - 277560*x^11 -
236360*x^10 - 260552*x^9 - 219170*x^8 - 229076*x^7 - 135900*x^6 - 379947*x^5 -
431255*x^4 - 24832*x^3 + 389941*x^2 + 506513*x - 328584, 110221*x^255 +
61924*x^254 + 261568*x^253 + 181722*x^252 + 77508*x^251 - 490726*x^250 -
418369*x^249 + 236541*x^248 - 504245*x^247 - 502068*x^246 + 451555*x^245 -
204574*x^244 + 115534*x^243 - 465447*x^242 + 27704*x^241 + 73815*x^240 -
109517*x^239 - 80129*x^238 + 210560*x^237 + 184676*x^236 + 426012*x^235 +
5518*x^234 + 429264*x^233 - 37140*x^232 - 113843*x^231 + 197636*x^230 +
217390*x^229 + 82092*x^228 - 353673*x^227 - 421549*x^226 - 352536*x^225 +
114220*x^224 - 452811*x^223 + 505640*x^222 + 192684*x^221 - 519357*x^220 +
422606*x^219 + 89566*x^218 - 453748*x^217 - 467110*x^216 + 281058*x^215 -
165195*x^214 + 405849*x^213 - 153813*x^212 - 204031*x^211 + 419417*x^210 +
198126*x^209 + 72510*x^208 + 410805*x^207 - 343688*x^206 + 30977*x^205 +
329760*x^204 + 57053*x^203 - 461930*x^202 - 285057*x^201 - 328173*x^200 +
368901*x^199 + 213593*x^198 - 512476*x^197 + 387361*x^196 + 52576*x^195 +
24574*x^194 + 81608*x^193 - 81327*x^192 + 153892*x^191 - 381228*x^190 -
182588*x^189 - 376622*x^188 + 479979*x^187 - 394667*x^186 + 523543*x^185 +
263445*x^184 - 467483*x^183 - 411665*x^182 - 116476*x^181 + 439146*x^180 -
141505*x^179 - 341996*x^178 - 239445*x^177 + 386150*x^176 + 215772*x^175 -
435182*x^174 - 290463*x^173 - 441285*x^172 + 453699*x^171 + 110223*x^170 +
61931*x^169 + 261556*x^168 + 181715*x^167 + 77504*x^166 - 490722*x^165 -
418365*x^164 + 236544*x^163 - 504244*x^162 - 502057*x^161 + 451563*x^160 -
204564*x^159 + 115536*x^158 - 465444*x^157 + 27706*x^156 + 73814*x^155 -
```

```
109517*x^154 - 80123*x^153 + 210562*x^152 + 184666*x^151 + 426011*x^150 +
5520*x^149 + 429272*x^148 - 37137*x^147 - 113852*x^146 + 197633*x^145 +
217393*x^144 + 82093*x^143 - 353668*x^142 - 421546*x^141 - 352528*x^140 +
114227*x^139 - 452801*x^138 + 505642*x^137 + 192689*x^136 - 519352*x^135 +
422611*x^134 + 89566*x^133 - 453737*x^132 - 467109*x^131 + 281058*x^130 -
165205*x^129 + 405847*x^128 - 153812*x^127 - 204032*x^126 + 419419*x^125 +
198132*x^124 + 72501*x^123 + 410816*x^122 - 343686*x^121 + 30970*x^120 +
329746*x^119 + 57058*x^118 - 461940*x^117 - 285060*x^116 - 328169*x^115 +
368903*x^114 + 213596*x^113 - 512475*x^112 + 387357*x^111 + 52577*x^110 +
24582*x^109 + 81615*x^108 - 81325*x^107 + 153885*x^106 - 381220*x^105 -
182597*x^104 - 376623*x^103 + 479977*x^102 - 394671*x^101 + 523544*x^100 +
263451*x^99 - 467478*x^98 - 411663*x^97 - 116479*x^96 + 439139*x^95 -
141511*x^94 - 341996*x^93 - 239443*x^92 + 386149*x^91 + 215773*x^90 -
435172*x^89 - 290456*x^88 - 441281*x^87 + 453704*x^86 + 110225*x^85 + 61924*x^84
+ 261554*x^83 + 181717*x^82 + 77505*x^81 - 490723*x^80 - 418368*x^79 +
236551*x^78 - 504242*x^77 - 502052*x^76 + 451564*x^75 - 204567*x^74 +
115533*x^73 - 465443*x^72 + 27702*x^71 + 73813*x^70 - 109521*x^69 - 80122*x^68 +
210562*x^67 + 184682*x^66 + 426007*x^65 + 5520*x^64 + 429272*x^63 - 37142*x^62 -
113847*x^61 + 197635*x^60 + 217386*x^59 + 82099*x^58 - 353669*x^57 - 421544*x^56
- 352527*x^55 + 114226*x^54 - 452802*x^53 + 505639*x^52 + 192685*x^51 -
519353*x^50 + 422611*x^49 + 89569*x^48 - 453738*x^47 - 467114*x^46 + 281063*x^45
- 165196*x^44 + 405852*x^43 - 153810*x^42 - 204035*x^41 + 419411*x^40 +
198128*x^39 + 72507*x^38 + 410806*x^37 - 343689*x^36 + 30978*x^35 + 329752*x^34
+ 57055*x^33 - 461927*x^32 - 285053*x^31 - 328178*x^30 + 368899*x^29 +
213588*x^28 - 512479*x^27 + 387354*x^26 + 52572*x^25 + 24574*x^24 + 81615*x^23 -
81327*x^22 + 153894*x^21 - 381224*x^20 - 182599*x^19 - 376622*x^18 + 479980*x^17
- 394674*x^16 + 523547*x^15 + 263452*x^14 - 467485*x^13 - 411658*x^12 -
116474*x^11 + 439140*x^10 - 141511*x^9 - 342001*x^8 - 239439*x^7 + 386148*x^6 +
215769*x^5 - 435178*x^4 - 290458*x^3 - 441283*x^2 + 453700*x + 110226,
-238288*x^255 - 437572*x^254 - 492045*x^253 - 61998*x^252 - 260778*x^251 -
294912*x^250 + 282590*x^249 + 72746*x^248 + 105615*x^247 - 8878*x^246 -
263406*x^245 - 367110*x^244 - 418078*x^243 - 492333*x^242 + 436701*x^241 -
145284*x^240 - 492455*x^239 - 234844*x^238 + 438860*x^237 - 103934*x^236 +
476320*x^235 + 378107*x^234 + 82293*x^233 + 351072*x^232 - 283289*x^231 -
93831*x^230 - 471438*x^229 + 45912*x^228 - 275899*x^227 - 89001*x^226 +
267165*x^225 - 186229*x^224 - 191869*x^223 + 258413*x^222 - 478422*x^221 +
110063*x^220 + 466254*x^219 + 522444*x^218 - 469242*x^217 - 412620*x^216 -
78798*x^215 + 192958*x^214 - 422684*x^213 + 126169*x^212 - 365978*x^211 -
373741*x^210 - 84834*x^209 + 5588*x^208 + 75354*x^207 - 448752*x^206 +
457093*x^205 + 167480*x^204 + 493192*x^203 + 229809*x^202 + 81865*x^201 -
192288*x^200 - 407904*x^199 - 49210*x^198 + 114271*x^197 - 274116*x^196 +
427066*x^195 + 488498*x^194 + 484771*x^193 + 297702*x^192 - 87399*x^191 -
300658*x^190 + 281208*x^189 - 185852*x^188 - 121093*x^187 + 233234*x^186 +
250695*x^185 - 125006*x^184 - 498899*x^183 - 179407*x^182 - 84222*x^181 +
250914*x^180 + 16551*x^179 - 388110*x^178 + 198146*x^177 + 16371*x^176 +
414889*x^175 - 78482*x^174 - 522661*x^173 - 445390*x^172 + 178645*x^171 -
238281*x^170 - 437576*x^169 - 492048*x^168 - 62004*x^167 - 260779*x^166 -
294907*x^165 + 282589*x^164 + 72742*x^163 + 105610*x^162 - 8878*x^161 -
```

```
263404*x^160 - 367111*x^159 - 418074*x^158 - 492327*x^157 + 436714*x^156 -
145288*x^155 - 492450*x^154 - 234845*x^153 + 438866*x^152 - 103933*x^151 +
476328*x^150 + 378105*x^149 + 82297*x^148 + 351071*x^147 - 283294*x^146 -
93836*x^145 - 471428*x^144 + 45916*x^143 - 275894*x^142 - 89001*x^141 +
267172*x^140 - 186227*x^139 - 191870*x^138 + 258418*x^137 - 478414*x^136 +
110074*x^135 + 466265*x^134 + 522448*x^133 - 469248*x^132 - 412621*x^131 -
78802*x^130 + 192954*x^129 - 422688*x^128 + 126185*x^127 - 365972*x^126 -
373733*x^125 - 84829*x^124 + 5595*x^123 + 75356*x^122 - 448749*x^121 +
457088*x^120 + 167475*x^119 + 493184*x^118 + 229815*x^117 + 81851*x^116 -
192297*x^115 - 407900*x^114 - 49202*x^113 + 114268*x^112 - 274103*x^111 +
427070*x^110 + 488501*x^109 + 484771*x^108 + 297697*x^107 - 87409*x^106 -
300651*x^105 + 281209*x^104 - 185853*x^103 - 121088*x^102 + 233247*x^101 +
250699*x^100 - 125006*x^99 - 498904*x^98 - 179404*x^97 - 84227*x^96 +
250916*x^95 + 16542*x^94 - 388108*x^93 + 198151*x^92 + 16368*x^91 + 414896*x^90
- 78478*x^89 - 522658*x^88 - 445386*x^87 + 178639*x^86 - 238301*x^85 -
437577*x^84 - 492059*x^83 - 62009*x^82 - 260781*x^81 - 294901*x^80 + 282598*x^79
+ 72750*x^78 + 105616*x^77 - 8878*x^76 - 263405*x^75 - 367109*x^74 - 418079*x^73
- 492332*x^72 + 436707*x^71 - 145284*x^70 - 492456*x^69 - 234843*x^68 +
438866*x^67 - 103928*x^66 + 476326*x^65 + 378109*x^64 + 82294*x^63 + 351068*x^62
- 283301*x^61 - 93833*x^60 - 471432*x^59 + 45915*x^58 - 275889*x^57 - 88991*x^56
+ 267171*x^55 - 186223*x^54 - 191869*x^53 + 258408*x^52 - 478421*x^51 +
110077*x^50 + 466266*x^49 + 522447*x^48 - 469235*x^47 - 412617*x^46 - 78799*x^45
+ 192952*x^44 - 422678*x^43 + 126179*x^42 - 365975*x^41 - 373736*x^40 -
84833*x^39 + 5590*x^38 + 75361*x^37 - 448755*x^36 + 457095*x^35 + 167484*x^34 +
493192*x^33 + 229812*x^32 + 81854*x^31 - 192297*x^30 - 407902*x^29 - 49210*x^28
+ 114272*x^27 - 274110*x^26 + 427075*x^25 + 488505*x^24 + 484773*x^23 +
297696*x^22 - 87402*x^21 - 300654*x^20 + 281211*x^19 - 185862*x^18 - 121092*x^17
+ 233247*x^16 + 250700*x^15 - 125010*x^14 - 498905*x^13 - 179406*x^12 -
84221*x^11 + 250921*x^10 + 16548*x^9 - 388110*x^8 + 198148*x^7 + 16369*x^6 +
414886*x^5 - 78486*x^4 - 522652*x^3 - 445390*x^2 + 178640*x - 238290,
-427314*x^255 + 157152*x^254 + 36983*x^253 - 428523*x^252 - 294211*x^251 +
234095*x^250 - 491544*x^249 + 298356*x^248 + 109305*x^247 - 109170*x^246 +
457187*x^245 - 330071*x^244 - 222712*x^243 + 305172*x^242 - 83565*x^241 -
8363*x^240 + 152060*x^239 + 121211*x^238 - 259574*x^237 + 348256*x^236 +
217460*x^235 - 260609*x^234 - 11230*x^233 + 476423*x^232 + 109579*x^231 +
174355*x^230 + 511583*x^229 + 146763*x^228 + 379191*x^227 + 314215*x^226 -
495954*x^225 + 510728*x^224 - 142143*x^223 - 8580*x^222 + 438337*x^221 +
298678*x^220 + 148174*x^219 + 403906*x^218 - 55778*x^217 + 292235*x^216 +
233553*x^215 - 437874*x^214 + 293618*x^213 + 173930*x^212 - 240681*x^211 +
442096*x^210 - 44593*x^209 + 463466*x^208 + 367875*x^207 - 354290*x^206 -
468406*x^205 + 507308*x^204 - 68018*x^203 - 403162*x^202 + 295142*x^201 -
487786*x^200 + 93025*x^199 - 292298*x^198 + 247942*x^197 - 89253*x^196 -
479658*x^195 - 296481*x^194 - 38125*x^193 + 82421*x^192 + 90919*x^191 -
138770*x^190 + 2117*x^189 - 249531*x^188 + 133657*x^187 - 341242*x^186 +
9675*x^185 + 276307*x^184 + 339180*x^183 - 217570*x^182 - 199868*x^181 -
27385*x^180 - 79630*x^179 - 167050*x^178 + 262982*x^177 + 494144*x^176 +
196781*x^175 - 21705*x^174 + 53578*x^173 - 301187*x^172 - 383785*x^171 -
427314*x^170 + 157147*x^169 + 36983*x^168 - 428520*x^167 - 294209*x^166 +
```

```
234090*x^165 - 491544*x^164 + 298360*x^163 + 109299*x^162 - 109173*x^161 +
457191*x^160 - 330065*x^159 - 222707*x^158 + 305180*x^157 - 83570*x^156 -
8357*x^155 + 152072*x^154 + 121212*x^153 - 259569*x^152 + 348261*x^151 +
217462*x^150 - 260614*x^149 - 11228*x^148 + 476415*x^147 + 109578*x^146 +
174358*x^145 + 511582*x^144 + 146773*x^143 + 379203*x^142 + 314216*x^141 -
495969*x^140 + 510735*x^139 - 142142*x^138 - 8581*x^137 + 438343*x^136 +
298689*x^135 + 148177*x^134 + 403913*x^133 - 55761*x^132 + 292234*x^131 +
233558*x^130 - 437871*x^129 + 293617*x^128 + 173930*x^127 - 240674*x^126 +
442096*x^125 - 44605*x^124 + 463469*x^123 + 367888*x^122 - 354286*x^121 -
468400*x^120 + 507312*x^119 - 68025*x^118 - 403163*x^117 + 295144*x^116 -
487790*x^115 + 93019*x^114 - 292294*x^113 + 247937*x^112 - 89259*x^111 -
479647*x^110 - 296478*x^109 - 38123*x^108 + 82422*x^107 + 90918*x^106 -
138769*x^105 + 2120*x^104 - 249540*x^103 + 133659*x^102 - 341240*x^101 +
9681*x^100 + 276315*x^99 + 339179*x^98 - 217570*x^97 - 199868*x^96 - 27386*x^95
- 79635*x^94 - 167047*x^93 + 262975*x^92 + 494137*x^91 + 196785*x^90 -
21700*x^89 + 53586*x^88 - 301183*x^87 - 383792*x^86 - 427314*x^85 + 157144*x^84
+ 36979*x^83 - 428517*x^82 - 294221*x^81 + 234091*x^80 - 491540*x^79 +
298356*x^78 + 109300*x^77 - 109175*x^76 + 457183*x^75 - 330069*x^74 -
222710*x^73 + 305180*x^72 - 83567*x^71 - 8361*x^70 + 152065*x^69 + 121216*x^68 -
259567*x^67 + 348260*x^66 + 217463*x^65 - 260617*x^64 - 11236*x^63 + 476416*x^62
+ 109573*x^61 + 174349*x^60 + 511580*x^59 + 146770*x^58 + 379203*x^57 +
314211*x^56 - 495955*x^55 + 510737*x^54 - 142143*x^53 - 8575*x^52 + 438338*x^51
+ 298682*x^50 + 148184*x^49 + 403905*x^48 - 55774*x^47 + 292239*x^46 +
233564*x^45 - 437873*x^44 + 293621*x^43 + 173936*x^42 - 240669*x^41 +
442100*x^40 - 44589*x^39 + 463457*x^38 + 367880*x^37 - 354286*x^36 - 468397*x^35
+ 507306*x^34 - 68019*x^33 - 403165*x^32 + 295144*x^31 - 487791*x^30 +
93019*x^29 - 292302*x^28 + 247945*x^27 - 89252*x^26 - 479650*x^25 - 296477*x^24
- 38115*x^23 + 82422*x^22 + 90916*x^21 - 138770*x^20 + 2115*x^19 - 249530*x^18 +
133663*x^17 - 341248*x^16 + 9678*x^15 + 276321*x^14 + 339178*x^13 - 217572*x^12
- 199869*x^11 - 27386*x^10 - 79617*x^9 - 167039*x^8 + 262972*x^7 + 494134*x^6 +
196786*x^5 - 21706*x^4 + 53574*x^3 - 301189*x^2 - 383795*x - 427313,
366880*x^255 - 37940*x^254 - 322309*x^253 - 399644*x^252 + 288898*x^251 -
350385*x^250 + 351609*x^249 - 479923*x^248 - 11488*x^247 - 438827*x^246 +
268405*x^245 + 329555*x^244 - 198313*x^243 + 256851*x^242 + 476715*x^241 -
122003*x^240 - 18720*x^239 - 327579*x^238 + 415713*x^237 + 69585*x^236 -
328441*x^235 + 218554*x^234 + 264936*x^233 + 128225*x^232 - 31611*x^231 -
172859*x^230 + 503416*x^229 - 480629*x^228 - 444757*x^227 - 300395*x^226 +
313720*x^225 + 79524*x^224 - 222095*x^223 - 50107*x^222 - 211031*x^221 -
384812*x^220 - 35445*x^219 + 278450*x^218 - 434073*x^217 + 252299*x^216 -
297925*x^215 - 360685*x^214 - 271116*x^213 + 325855*x^212 + 438012*x^211 -
403763*x^210 - 43094*x^209 + 8621*x^208 + 59654*x^207 - 487505*x^206 +
118619*x^205 - 298672*x^204 + 513451*x^203 + 187608*x^202 + 471041*x^201 -
430726*x^200 + 218050*x^199 - 67861*x^198 - 264726*x^197 + 329901*x^196 -
203604*x^195 - 518326*x^194 - 407248*x^193 + 331320*x^192 - 257674*x^191 +
518564*x^190 - 112589*x^189 - 212599*x^188 + 183358*x^187 - 71233*x^186 -
127204*x^185 + 362465*x^184 + 128630*x^183 + 421303*x^182 - 251245*x^181 +
85792*x^180 - 193866*x^179 - 397764*x^178 + 506309*x^177 - 226247*x^176 +
445610*x^175 + 455124*x^174 - 495520*x^173 + 267445*x^172 - 36258*x^171 +
```

```
366877*x^170 - 37936*x^169 - 322312*x^168 - 399653*x^167 + 288889*x^166 -
350384*x^165 + 351611*x^164 - 479928*x^163 - 11499*x^162 - 438821*x^161 +
268411*x^160 + 329568*x^159 - 198312*x^158 + 256856*x^157 + 476724*x^156 -
121989*x^155 - 18717*x^154 - 327585*x^153 + 415715*x^152 + 69591*x^151 -
328448*x^150 + 218546*x^149 + 264941*x^148 + 128229*x^147 - 31612*x^146 -
172865*x^145 + 503410*x^144 - 480623*x^143 - 444747*x^142 - 300396*x^141 +
313720*x^140 + 79531*x^139 - 222090*x^138 - 50109*x^137 - 211033*x^136 -
384813*x^135 - 35436*x^134 + 278451*x^133 - 434059*x^132 + 252308*x^131 -
297908*x^130 - 360680*x^129 - 271114*x^128 + 325850*x^127 + 438021*x^126 -
403758*x^125 - 43094*x^124 + 8611*x^123 + 59651*x^122 - 487501*x^121 +
118618*x^120 - 298680*x^119 + 513456*x^118 + 187607*x^117 + 471031*x^116 -
430729*x^115 + 218053*x^114 - 67867*x^113 - 264717*x^112 + 329898*x^111 -
203605*x^110 - 518317*x^109 - 407237*x^108 + 331307*x^107 - 257673*x^106 +
518568*x^105 - 112587*x^104 - 212595*x^103 + 183351*x^102 - 71234*x^101 -
127195*x^100 + 362466*x^99 + 128628*x^98 + 421304*x^97 - 251245*x^96 +
85797*x^95 - 193866*x^94 - 397765*x^93 + 506316*x^92 - 226251*x^91 + 445602*x^90
+ 455121*x^89 - 495519*x^88 + 267447*x^87 - 36253*x^86 + 366877*x^85 -
37939*x^84 - 322310*x^83 - 399647*x^82 + 288895*x^81 - 350388*x^80 + 351614*x^79
- 479923*x^78 - 11500*x^77 - 438821*x^76 + 268406*x^75 + 329558*x^74 -
198308*x^73 + 256854*x^72 + 476725*x^71 - 121992*x^70 - 18714*x^69 - 327579*x^68
+ 415714*x^67 + 69592*x^66 - 328447*x^65 + 218546*x^64 + 264934*x^63 +
128227*x^62 - 31619*x^61 - 172859*x^60 + 503409*x^59 - 480623*x^58 - 444750*x^57
- 300390*x^56 + 313722*x^55 + 79529*x^54 - 222094*x^53 - 50106*x^52 -
211025*x^51 - 384813*x^50 - 35439*x^49 + 278458*x^48 - 434066*x^47 + 252312*x^46
- 297904*x^45 - 360677*x^44 - 271114*x^43 + 325853*x^42 + 438015*x^41 -
403765*x^40 - 43096*x^39 + 8619*x^38 + 59655*x^37 - 487501*x^36 + 118621*x^35 -
298674*x^34 + 513455*x^33 + 187614*x^32 + 471035*x^31 - 430728*x^30 +
218058*x^29 - 67861*x^28 - 264728*x^27 + 329898*x^26 - 203603*x^25 - 518321*x^24
- 407230*x^23 + 331314*x^22 - 257677*x^21 + 518573*x^20 - 112588*x^19 -
212604*x^18 + 183358*x^17 - 71233*x^16 - 127197*x^15 + 362471*x^14 + 128628*x^13
+ 421306*x^12 - 251240*x^11 + 85793*x^10 - 193865*x^9 - 397763*x^8 + 506315*x^7
- 226244*x^6 + 445602*x^5 + 455117*x^4 - 495520*x^3 + 267443*x^2 - 36248*x +
366878), [False, False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, True, True, False,
False, True, False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False, False, False,
False, False, True, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, True, False, False, False,
False, False, False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False, False, False,
```

```
False, False, False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, True, False, False, True, False,
False, False, False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False, False, False,
False, False, False, True, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, True, False, False, False,
False, False, False, False, False, False, False, False, False, False, False,
False, False, False, False, True, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, True, False, False, False, False,
False, True, False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, True, False, False, False, False,
False, False, False, False, True, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False, False, False,
False, False, False, True, False, False, False, False, False, False, False,
False, False, False, False, True, True, False, False, False, False, False,
False, False, False, False, False, False, False, False, False, False, False,
False, True, False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False, False, False,
True, False, False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, True, False, False, False,
False, False, False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, True, False, True, False,
False, False, False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False, False, False,
```

False, False, False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, True, False, False, False,
False, False, False, False, False, False, True, False, True, False, False,
False, False, False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False, False, False,
False, False, False, True, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False, False, True,
False, False, False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False, False, False,
False, False, True, False, False, False, False, True, False, False, False,
False, False, False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False, False, False,
False, False, False, True, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False, False, False,
False, True, False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False, False, False,

False, True, False, False, False, False, False, False, False, False, False,
False, False, True, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, True, False, False, False,
False, False, False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False, False, False,
False, False, True, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, True, False, False,
False, False, False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, True, False, False, False, False,
False, True, False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False, False, False,
False, True, False, False, False, True, False, False, False, False, False,
False, False, False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, True, False, False,
False, False, False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, True, False, False,
False, False, False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, True, False, False, True,
False, False, False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False, False, False,

```
False, False, False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, True, True, False, False, False,
False, False, False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, True, False, False, False, False,
False, False, False, False, False, False, False, False, False, True, False,
False, False, False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False, False, False,
False, False, False, True, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False, False, False,
False, False, False, False, True, False, False, True, False, False, False,
False, False, False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False, False, False,
False, False, False, False, False, True, False, False, False, False, False,
False, False, False, False, False, False, False, False, False, False, False,
True, False, False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, True, False, False, False, False,
False, False, False, False, False, True, False, False, False, False, False,
False, False, False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False, True, False,
False, False, False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False, False, False, False, False, False,
```

```
False, False, False, False, False, False, False, False, False, False, False,
False, False, False, False, False, False], b'\x87wo\xcc\x8eA\x01p?H7\x1cw\xea?\n
\xf2\x81\xb7|\xd4\xb9E\xba\x9e{\x1b\x15\x04\xeb*\x98')
MENSAGEM VÁLIDA
Mensagem a ser assinada
Assinatura Válida!
MENSAGEM INVÁLIDA
Mensagem a ser assinado
Assinatura Inválida!
```