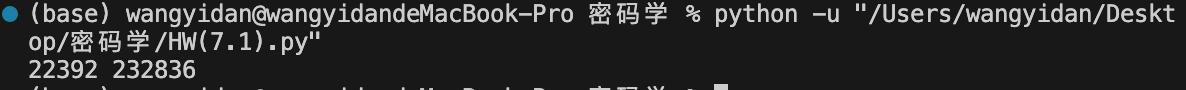
7.1

|  |
| --- |
| import math  def shanks\_algorithm(alpha, beta, p):  # Step 1: Initialize  m = math.ceil(math.sqrt(p))  # Step 2: Baby-step  baby\_steps = {}  for i in range(m):  value = pow(alpha, i, p)  baby\_steps[value] = i  # Step 3: Giant-step  inv\_alpha\_m = pow(alpha, -m, p)  giant\_step = beta  for j in range(m):  if giant\_step in baby\_steps:  i = baby\_steps[giant\_step]  return j \* m + i  giant\_step = (giant\_step \* inv\_alpha\_m) % p  return None  # Test cases from the problem  alpha1, beta1, p1 = 106, 12375, 24691  alpha2, beta2, p2 = 6, 248388, 458009  log1 = shanks\_algorithm(alpha1, beta1, p1)  log2 = shanks\_algorithm(alpha2, beta2, p2)  print(log1, log2) |

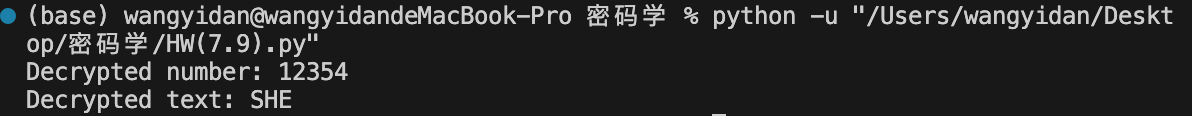
Result:(解释部分在文字）



7.9

|  |
| --- |
| import sympy  # 定义参数  p = 31847  alpha = 5  a = 7899  # 定义一个函数进行解密  def elgamal\_decrypt(c1, c2, p, a):  s = pow(c1, a, p)  s\_inv = sympy.mod\_inverse(s, p)  m = (c2 \* s\_inv) % p  return m  # 示例密文对  ciphertext\_pairs = [(3781, 14409)]  # 解密示例  for c1, c2 in ciphertext\_pairs:  m = elgamal\_decrypt(c1, c2, p, a)  print(f"Decrypted number: {m}")  # 解码为字母  first\_letter = chr((m // (26\*\*2)) + ord('A'))  second\_letter = chr(((m // 26) % 26) + ord('A'))  third\_letter = chr((m % 26) + ord('A'))  print(f"Decrypted text: {first\_letter}{second\_letter}{third\_letter}") |

Result:（解释部分在文字）



7.23

|  |
| --- |
| import random  class FiniteGroup:  def \_\_init\_\_(self, order):  self.\_order = order  self.\_generator = random.randint(2, order - 1)  def order(self):  return self.\_order  def generator(self):  return self.\_generator  def random\_element(self):  return random.randint(2, self.\_order - 1)  class OracleDDH:  def \_\_init\_\_(self, G):  self.G = G  def is\_DDH\_tuple(self, g, g\_a, g\_b, g\_ab):  # Simulated DDH oracle logic for this example  a = pow(g\_a, 1, self.G.order())  b = pow(g\_b, 1, self.G.order())  return pow(g, a \* b, self.G.order()) == g\_ab  class ElGamal:  def \_\_init\_\_(self, G, alpha, beta):  self.G = G  self.alpha = alpha  self.beta = beta  def encrypt(self, x):  k = random.randint(1, self.G.order() - 1)  y1 = pow(self.alpha, k, self.G.order())  y2 = (x \* pow(self.beta, k, self.G.order())) % self.G.order()  return (y1, y2)  class OracleDISTINGUISH:  def \_\_init\_\_(self, G):  self.G = G  def distinguish(self, x1, x2, y1, y2):  # Simulated distinguishing logic for this example  return y2 in [(x1 \* y1) % self.G.order(), (x2 \* y1) % self.G.order()]  def distinguish\_ElGamal\_encryptions(x1, x2, y1, y2, oracle\_ddh):  g = oracle\_ddh.G.generator()  beta = oracle\_ddh.G.random\_element()  is\_DDH\_1 = oracle\_ddh.is\_DDH\_tuple(g, y1, beta, y2 \* pow(x1, -1, oracle\_ddh.G.order()))  is\_DDH\_2 = oracle\_ddh.is\_DDH\_tuple(g, y1, beta, y2 \* pow(x2, -1, oracle\_ddh.G.order()))  if is\_DDH\_1:  return True  elif is\_DDH\_2:  return True  else:  return False  def solve\_DDH(g, g\_a, g\_b, g\_ab, oracle\_distinguish):  x1 = 1  x2 = 2  y1 = g\_a  y2\_1 = g\_ab  y2\_2 = (g\_ab \* pow(g\_a, -1, oracle\_distinguish.G.order())) % oracle\_distinguish.G.order()  if oracle\_distinguish.distinguish(x1, x2, y1, y2\_1):  return True  elif oracle\_distinguish.distinguish(x1, x2, y1, y2\_2):  return False  else:  return False  # Example usage  order = 1019 # A prime number for the order of the group  G = FiniteGroup(order)  alpha = G.generator()  beta = G.random\_element()  oracle\_ddh = OracleDDH(G)  elgamal = ElGamal(G, alpha, beta)  x1 = random.randint(1, order - 1)  x2 = random.randint(1, order - 1)  y1, y2 = elgamal.encrypt(x1)  result\_distinguish = distinguish\_ElGamal\_encryptions(x1, x2, y1, y2, oracle\_ddh)  print(f"ElGamal encryption distinguishing result: {result\_distinguish}")  oracle\_distinguish = OracleDISTINGUISH(G)  g = G.generator()  a = random.randint(1, order - 1)  b = random.randint(1, order - 1)  g\_a = pow(g, a, G.order())  g\_b = pow(g, b, G.order())  g\_ab = pow(g, a \* b, G.order())  result\_solve\_ddh = solve\_DDH(g, g\_a, g\_b, g\_ab, oracle\_distinguish)  print(f"Decision Diffie-Hellman result: {result\_solve\_ddh}") |

Result:(解释部分在文字）

