一、实验目标：

在本次实验中我们使用泛化哈希链实现验证一个秘密数 secret 在某个区间[a,b]内。

二、实验流程：

选择一个合适的区间[a, b]，表示想要证明的秘密整数 x 的范围。选择一个合适的哈希函数族{H_k | k 是任意整数}

需要生成一个初始值 seed，，它可以是任意的字符串或数字。计算出 x 在区间[a, b]中对应的二进制表示 b_x，并且将其分成 n 个比特位 b_x[0], b_x1, …, b_x[n-1]。

根据 b_x 中每个比特位的值来选择不同的参数 k，并且用它们来构造一个泛化哈希链。具体地说，如果 b_x[i]=0，那么您就选择 k=2i；如果 b_x[i]=1，选择 k=-2i。然后，可以用这些参数 k 来生成 n 个哈希值 h_0, h_1, …, h_n-1，其中 h_i = H_k(s)。如果 b_x=101010，并且 s=“hello”，那么 h_0 = SHA-256–1(“hello”)；h_1 = SHA-256–2(“hello”)；h_2 = SHA-256-4(“hello”)；h_3 = SHA-256–8(“hello”)；h_4 = SHA-256-16(“hello”)；h_5 = SHA-256–32(“hello”)。

将种子 s 和 n 个哈希值 h_0, h_1, …, h_n-1 组成一个证明 p，并且将其发送给验证者。验证者可以根据以下方法来验证证明 p：

首先，验证者需要知道区间[a, b]和哈希函数族{H_k | k 是任意整数}。

然后，验证者需要从证明 p 中提取出种子 s 和 n 个哈希值 h_0, h_1, …, h_n-1。

接下来，验证者需要根据哈希值 h_0, h_1, …, h_n-1 来重构出 b_x 中每个比特位的值。如果 h_i = H_k(s)，那么验证者就可以根据 k 的正负号来判断 b_x[i]的值。如果 k>0，那么 b_x[i]=0；如果 k<0，那么 b_x[i]=1。

最后，验证者需要根据 b_x 中的二进制表示来计算出 x 的值，并且检查是否满足 x 在区间[a, b]中。

三、实验操作：

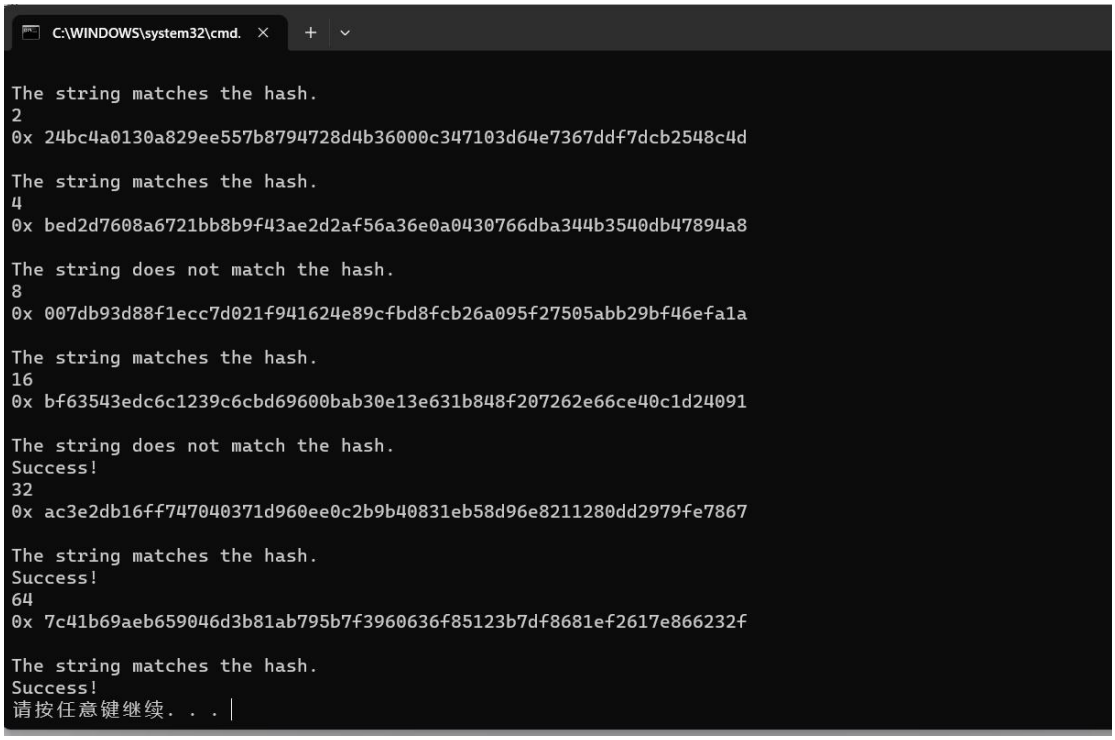
在实验中我们选择了 python 自带的 hashlib 库中的 SHA-256 作为哈希函数。通过将 seed 与二进制数 k 相异或得到一系列的哈希函数族。

```
def hash_encode(s,k):
    # 创建一个 SHA-256 哈希对象
    m = hashlib.sha256()
        # 向哈希对象中添加数据
    m.update(bxor(s,k).encode())
    # 获取哈希值
    h = m.digest()
    # 将哈希值转换为 16 进制字符串
    d = m.hexdigest()
    #打印哈希值的 16 进制
    print("0x",d,"\n")
    return d
```

生成 b_x 时我们采用 srcert 在区间[a,b]的相对位置的二进制，并取 log(b,2)的下界为位数 n，表示要表示 b_x 需要的位数。

之后生成一系列的哈希链。

在比较时我们选择比较正确返回’0’，失败则返回’1’，从而实现对 b_x 的还原。要注意的是，此时验证者并不能知道 secret 具体是哪一个数。



附录：

```
#coding=gbk
import hashlib
import math
def stb(string):
    bytes_obj = string.encode() # 转换为 bytes 类型
    #print(type(bytes_obj), bytes_obj)
    int_obj = int.from_bytes(bytes_obj, byteorder='big') # 转换为 int
类型
    #print(type(int_obj), int_obj)
    str_obj = bin(int_obj) # 转换为 str 类型
    #print(type(str_obj), str_obj)
    bin_str = str_obj.replace('0b', '') # 去掉前缀 '0b'
    str = bin_str.ljust(256, '0') # 在右侧补足 0
    #print( bin_str)
    return str

#哈希函数_参数k用来表示哈希族
def hash_encode(s,k):
    # 创建一个SHA-256哈希对象
```

```
    m = hashlib.sha256()
        # 向哈希对象中添加数据
    m.update(bxor(s,k).encode())
    # 获取哈希值
    h = m.digest()
    # 将哈希值转换为16进制字符串
    d = m.hexdigest()
    #打印哈希值的16进制
    print("0x",d,"\n")
    return d


#验证函数 0对 1错
def hash_verif(s,d_,k):
    d=hash_encode(stb(s),dtb(k))
    if d==d_:
        print("The string matches the hash.")
        return 0
    else:
        print("The string does not match the hash.")
        return 1
```

```
def dtb(num):
    if num>=0:
        str_obj = bin(num) #  转换为  str  类型
        #print(type(str_obj), str_obj)
        bin_str = str_obj.replace('0b', '') #  去掉前缀  '0b'
        bin_str = bin_str.rjust(256, '0') #  在左侧补足  0
    else:
        num=-num-1
        str_obj = bin(num) #  转换为  str  类型
        #print(type(str_obj), str_obj)
        bin_str = str_obj.replace('0b', '') #  去掉前缀  '0b'
        bin_str = bin_str.rjust(256, '0') #  在左侧补足  0
        bin_str = bin_str.replace('0', '3')
        bin_str = bin_str.replace('1', '0')
        bin_str = bin_str.replace('3', '1')
    #print( bin_str)
    return bin_str



def dec_to_binary(num,n):
    str_obj = bin(num)[2:] #  转换为  str  类型
```

```
    if len(str_obj)<n:
        str_obj=str_obj.rjust(n,'0')
    else:
        str_obj=str_obj[len(str_obj)-n:]
    print(str_obj)
    return str_obj
#对编码后的对象进行对齐的异或运算
def bxor(s,k):
    str1 = s
    str2 = k
    int1 = int(str1, 2) # 将第一个字符串转换为 int 类型
    int2 = int(str2, 2) # 将第二个字符串转换为 int 类型
    int3 = int1 ^ int2 # 对两个 int 类型进行异或运算
    #print(type(int3), int3)
    str3 = bin(int3)[2:] # 将 int 类型转换为 str 类型
    str3=str3.rjust(256,'0')
    #print(str3)
    return str3

def verif(s,h_i,n,num):
    p=[]
    verif_num=0
```

```
    for i in range(0,n):
        k=2**i
        print(k)
        a=hash_verif(s,h_i[i],k)
        verif_num+=a*2**(n-i-1)
        if num==verif_num:
            print("Success!")


#哈希链想要证明所给的数属于[1:100]，数为20
seed="Hello World!"
a=1
b=100
l=b-a+1
c=20
p=c-a+1
n=math.log(l,2)
if n>n//1:
    n=int(n//1)+1
b_x=dec_to_binary(p,n)
```

```
k_i=[]
h_i=[]
#得到h_i
for i in range(0,n):
    if b_x[i]=='0':
        k=2**i
    else:
        k=-2**i
    print(k)
    h_i.append(hash_encode(stb(seed),dtb(k)))
    k_i.append(k)

verif(seed,h_i,n,p)
```