**Exercise1**

The whole protocol contains 4 steps.

Step1:

Alice send Sa and N to Bob. Now we set Sa=8 and Sb=10.

Step2:

Bob needs to generate the (g, p, q), B and AUTHbob(also signature) where we set the signing message = g+p+q+B. Also, the g, p, q, B need to meet the requirement. (See in the choose\_gpq() function)

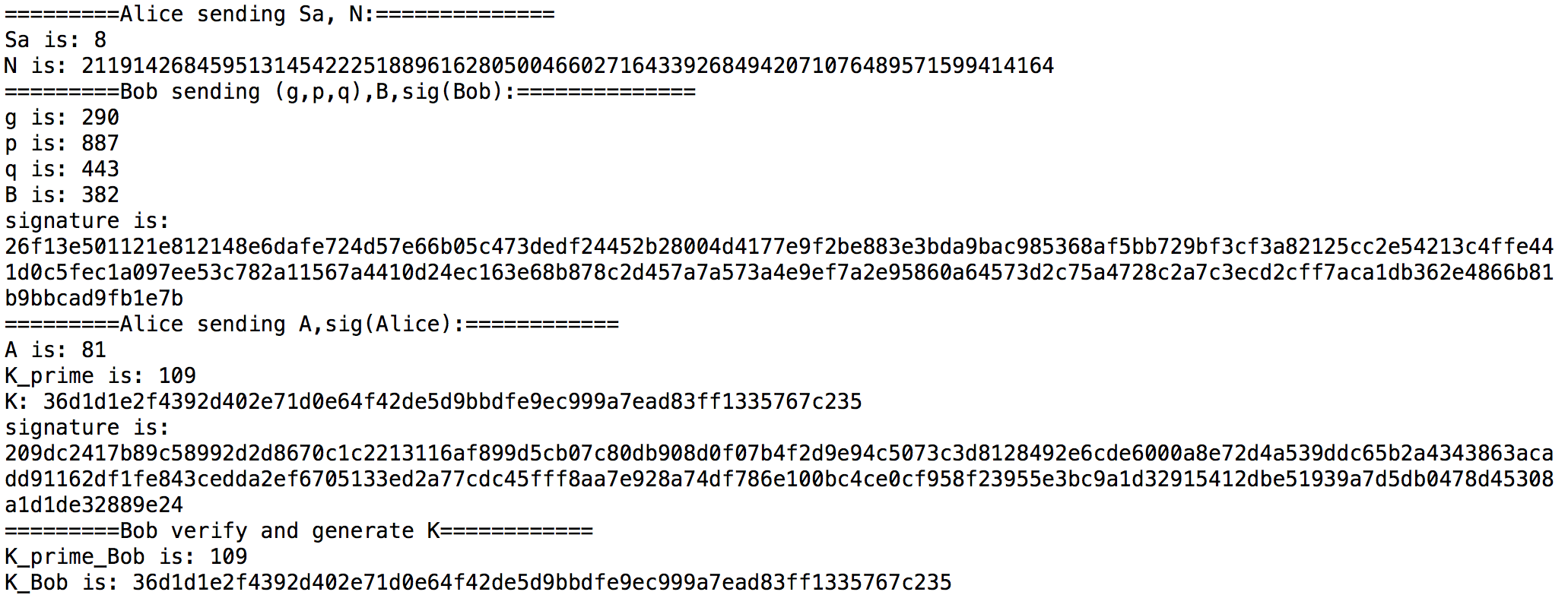
Step3:

Alice first verify the signature and a number of conditions. After that, Alice generate K’ and K, sending the A to Bob. Meanwhile, Alice generate the AUTHAlice(also signature) and send to Bob where we set the signing message is A. (See in the receiveFromAlice() function)

Step4:

Finally, Bob first verify the signature and a number of conditions. After that, Bob generate K’ and K (See in the bobReceive () function)

The result is shown in:



We can see the K generated from Alice and Bob are equal. The code is below:

|  |
| --- |
| import random  from Crypto.Hash import SHA256  from Crypto.Signature import PKCS1\_v1\_5  from Crypto.PublicKey import RSA  import math  def generatePrime(keysize):  while True:  num = random.randrange(2\*\*(keysize-1), 2\*\*(keysize))  if isPrime(num):  return num  def isPrime(n):  if n <=1: return False  i = 2  while i\*i <= n:  if n%i == 0: return False  i += 1  return True  def choose\_gpq(Sa, Sb):  alfa = 2  g = 0  s = max(Sa, Sb)  if s > 2\*Sb:  return False  while(True):  p = generatePrime(s)  if math.log(p,2) < s-1:  continue  q = (p-1)/2  if not isPrime(q):  continue  while(True):  alfa = random.randint(2, p-2)  g = alfa\*\*2 % p  if g == 1:  continue  if g == p-1:  continue  break  b = random.randint(1, q-1)  B = g\*\*b % p  break  message = g+p+q+B  key = RSA.generate(1024)  publickey = key.publickey()  h = SHA256.new(hex(message))  signature = PKCS1\_v1\_5.new(key).sign(h)  return g, p, q, b, B, alfa, publickey, signature  def receiveFromAlice(Sa, g, p, q, B, publickey, signature):  message = g+p+q+B  hash\_message = SHA256.new(hex(message))  if PKCS1\_v1\_5.new(publickey).verify(hash\_message, signature) == False:  print 'error'  if ((Sa-1) < math.log(p,2)) & (math.log(p,2) < 2\*Sa):  if not isPrime(q):  print 'error'  if not isPrime(p):  print 'error'  if 2\*q != (p-1):  print 'error'  if g == 1:  print 'error'  if g == p-1:  print 'error'  if B == 1:  print 'error'  if B\*\*q % p != 1:  print 'error'  a = random.randint(1, q-1)  A = g\*\*a % p  K\_prime = B\*\*a % p  h = SHA256.new()  h.update(hex(K\_prime))  K = h.hexdigest()  message1 = A  key\_new = RSA.generate(1024)  publickey\_new = key\_new.publickey()  hash\_new = SHA256.new(hex(message1))  signature\_new = PKCS1\_v1\_5.new(key\_new).sign(hash\_new)  return a, A, K\_prime, K, publickey\_new, signature\_new  else:  print 'error'    def bobReceive(A, g, p, q, b, publickey\_new, signature\_new):  message = A  hash\_message = SHA256.new(hex(message))  if PKCS1\_v1\_5.new(publickey\_new).verify(hash\_message, signature\_new) == False:  print 'error'  if A == 1:  print 'error'  if A\*\*q % p != 1:  print 'error'  K\_prime = A\*\*b % p  h = SHA256.new()  h.update(hex(K\_prime))  K = h.hexdigest()  return K\_prime, K    #  Sa = 8  Sb = 10  N = random.randint(0, 2\*\*256-1)  print '=========Alice sending Sa, N:=============='  print 'Sa is:', Sa  print 'N is:', N  print '=========Bob sending (g,p,q),B,sig(Bob):=============='  g, p, q, b, B, alfa, publickey, signature = choose\_gpq(Sa, Sb)  print 'g is:', g  print 'p is:', p  print 'q is:', q  print 'B is:', B  print 'signature is:', signature.encode('hex')  print '=========Alice sending A,sig(Alice):============'  a, A, K\_prime, K, publickey\_new, signature\_new = receiveFromAlice(Sa, g, p, q, B, publickey, signature)  print 'A is:', A  print 'K\_prime is:', K\_prime  print 'K:', K  print 'signature is:', signature\_new.encode('hex')  print '=========Bob verify and generate K============'  K\_prime\_FromBob, K\_FromBob = bobReceive(A, g, p, q, b, publickey\_new, signature\_new)  print 'K\_prime\_Bob is:', K\_prime\_FromBob  print 'K\_Bob is:', K\_FromBob |