```
################## Alice and Bob (Without Eve attack)
###################

import numpy as np


# The function that could generate primes in a range one
by one
def primes(start, stop):
    if start < 2:
        start = 2 # If the number is negative, 0, 1 or 2,
then choose 2 conveniently
    for i in range(start, stop+1): # Loop for generating
all primes in the range
        for j in range(2, i):
            if i % j == 0: # If i chould be divided by j,
then i is not a prime
                break
        else:
             yield i # Generating one of the primes in
the range and back to the beginning of the loop to repeat
untill no more prime


# The function that could generate factors of a non-prime
integer
def factor(n):
    fact = []
    i = 2
    while i<=n:
        if n%i==0:
            fact.append(i)
            n//= i
        else:
            i+=1
    return fact


# Get set of primes
i_start, i_stop = 0, 30 # Setting a range
x_set = [x for x in primes(i_start, i_stop)] # Collecting
primes into x_set
print('The set of primes from ' + str(i_start) + ' to ' +
str(i_stop) + ' is ' + str(x_set))
```

```
# Generating p and g
p = np.random.choice(x_set)

factor_set = list(set(factor(p-1)))
alpha_set = [] # Generating a null set for primitive root
for i in range(0, len(factor_set)):
    power_factor = (p-1)/factor_set[i]
    for alpha_factor in range(1, p):
        if alpha_factor^power_factor != 1:
            alpha_set = alpha_set + [alpha_factor]
g_set = list(set(alpha_set))
g = np.random.choice(g_set) # Random primitive root of p
print('The randomly choosed p and g from the set of
primes are ' + str(p) + ' and ' + str(g) + '
respectively.')


# Generating a and b
a = np.random.randint(1, p-1) # The Alice's secret
interger
b = np.random.randint(1, p-1) # The Bob's secret interger
while a == b: # a cannot equal to b
    a = np.random.randint(1, p-1)
    b = np.random.randint(1, p-1)
print('The randomly choosed a and b from '+ str(1) + ' to
' + str(p-1) + ' are ' + str(a) + ' and ' + str(b) + '
respectively. ')


# Result between Alice and Bob
A = int((g**a) % p) # The public key Alice send to Bob
B = int((g**b) % p) # The public key Bob send to Alice
while A == B:
    A = int((g**a) % p) # The public key Alice send to
Bob
    B = int((g**b) % p) # The public key Bob send to
Alice
K_a = int((B**a) % p) # Alice computes K_a = B^a mod p
K_b = int((A**b) % p) # Bob computes K_b = A^b mod p
print('The public key sent from Alice and Bob are ' +
str(A) + ' and ' + str(B) + ' respectively.')
print('The common secret key that Alice and Bob get are '
+ str(K_a) +' and '+ str(K_b) + ' respectively.')


# Judging whether Alice and Bob success
if K_a == K_b: # Alice and Bob share the same secret
number
    print('Alice and Bob now share a common secret key '
+ str(K_a) + ', so they success!')
else:
    print('Not true.')
```

```
The set of primes from 0 to 30 is [2, 3, 5, 7, 11, 13,
17, 19, 23, 29]
The randomly choosed p and g from the set of primes are
23 and 20 respectively.
The randomly choosed a and b from 1 to 22 are 2 and 14
respectively.
The public key sent from Alice and Bob are 9 and 4
respectively.
The common secret key that Alice and Bob get are 16 and
16 respectively.
Alice and Bob now share a common secret key 16, so they
success!
```

In [48]:

```
################## Eve (With attack) ###################
# Eve want to attack Alice and Bob

# Generating wrong key to attack Alice and Bob
c = np.random.randint(1, p-1) # Eve choose a number to
attack Alice and Bob
while c == a or c == b: # c cannot equal to a or b
    c = np.random.randint(1, p-1)



# Attacking
A_e = int(g**c % p)
B_e = int(g**c % p)
while A_e == 0 or B_e == 0:
    A_e = int(g**c % p)
    B_e = int(g**c % p)
print('Eve send the same wrong public key to Bob and
Alice , which is ' + str(A_e))
```

```
# Result after attacking
A_e_bob = int(A_e**b % p)
B_e_alice = int(B_e**a % p)
print('Alice and Bob calculate their wrong final secret
key which is ' + str(A_e_bob) + ' and ' + str(B_e_alice)
+ ' respectively.')


# Judging whether Eve success
if A_e_bob <> B_e_alice or A_e_bob <> K_a or B_e_alice <>
K_b:
    print('Eve success!')
```

```
Eve send the same wrong public key to Bob and Alice ,
which is 21
Alice and Bob calculate their wrong final secret key
which is 8 and 4 respectively.
Eve success!
```

In [49]:

```
##################### Eve (Calculate - Normal)
#######################
# Eve konws value of A, B, g, p and want to find out a,
b, and the final secret number


# Eve find a
for e_a in range(1, p-1):
    while int(g**e_a % int(p)) == A:
        if e_a == a:
            continue
        else: raise Exception('Wrong')
        break
    e_a = a
    break
print('Eve tried ' + str(e_a - 1) +' times to find out a,
which is ' + str(e_a))


# Eve find b
for e_b in range(1, p-1):
    while int(g**e_b % int(p)) == B:
        if e_b == b:
            continue
        else: raise Exception('Wrong')
        break
    e_b = b
    break
print('Eve tried ' + str(e_b - 1) +' times to find out b, 
which is ' + str(e_b))


e_s = int((g**e_b % p)**e_a % p)
e_s == int((g**e_a % p)**e_b % p)


# Eve find the secret number
e_s = int((g**e_b % p)**e_a % p) # Test whether is true
if e_s == int((g**e_a % p)**e_b % p):
    if e_s == K_b:
        print('Now Eve get the final secret number ' + 
str(e_s))
    else: raise Exception('Wrong')
else: raise Exception('Wrong')
```

```
Eve tried 1 times to find out a, which is 2
Eve tried 13 times to find out b, which is 14
Now Eve get the final secret number 16
```

In [50]:

```
##################### Eve (Calculate - Baby Step Giant 
Step Problem) #######################
# Eve konws value of A, B, g, p and want to find out a, 
b, and the final secret number

from math import ceil, sqrt

g = int(g)
# To solve h = g^x mod p = a and find x
def bsgs(g, h, p):
    m = int(ceil(sqrt(p-1)))
    # Baby Step
    lookup_table = {pow(g, i, p): i for i in 
range(int(m))}
    # Giant Step Pre-computation c = g^(-m) mod p 
(Fermat's Little Teorem)
    c = pow(g, m * (p - 2), p)
    # Giant Step
    for j in range(m):
        y = (h*pow(c, j, p)) % p
        if y in lookup_table:
            return j*m + lookup_table[y]
    return 'Nothing'


print('Eve found out the a, which is')
print(bsgs(g, A, p))
if bsgs(g, A, p) == a:
    print('Eve found out the a!')
else:
    print('The result of a is not true.')



print('Eve found out the b, which is')
print(bsgs(g, B, p))
if bsgs(g, B, p) == b:
    print('Eve found out the b!')
else:
    print('The result of b not true.')
```

```
Eve found out the a, which is
2
Eve found out the a!
Eve found out the b, which is
14
Eve found out the b!
```