Experiment 2

Diffie Hellman Algorithm

Name: Jash Jain
UID: 2019130021
Class: TE Comps

Objective:
To implement the Diffie Hellman Algorithm

Theory:

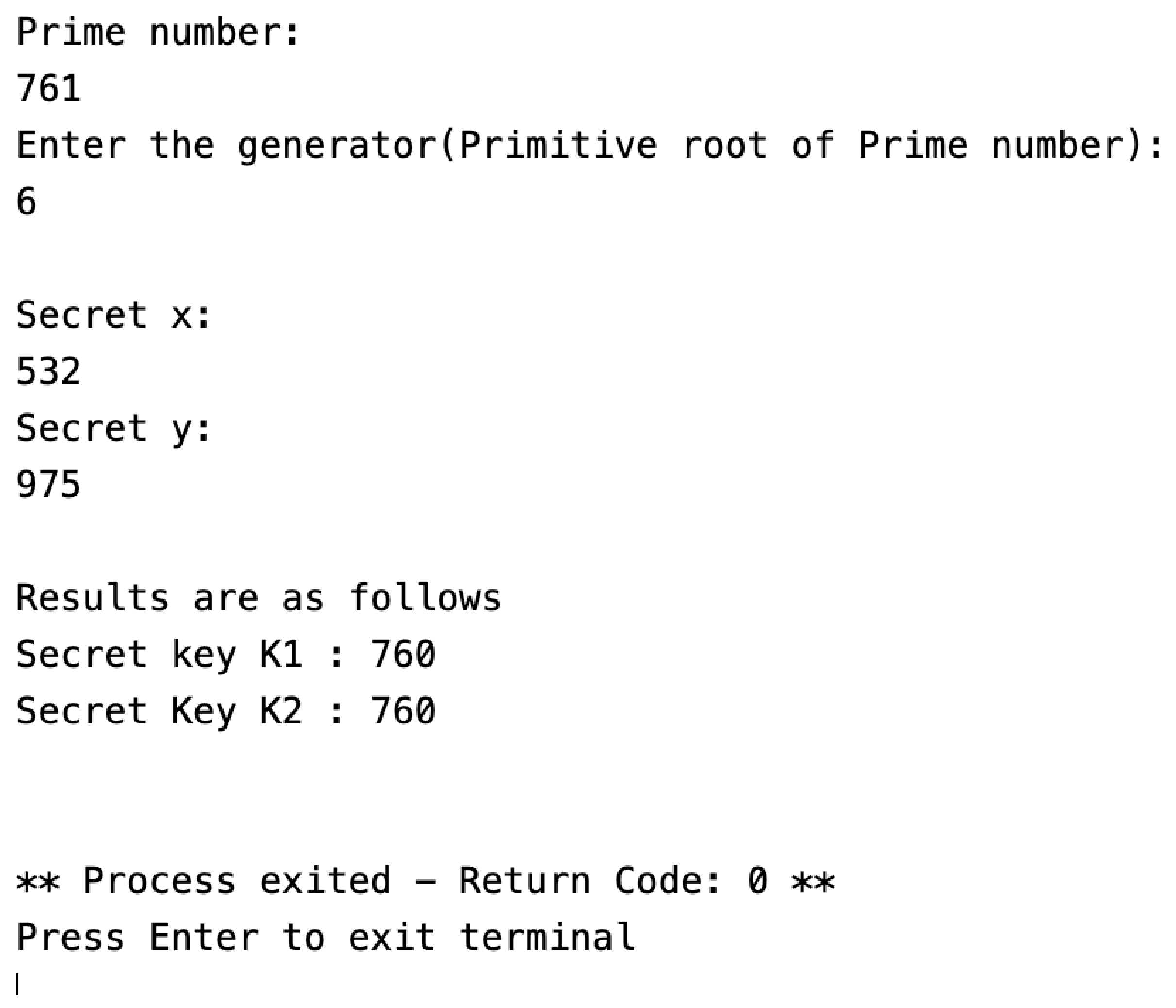
Diffie–Hellman key exchange is a method of securely exchanging cryptographic keys over a public channel and was one of the first public-key protocols as conceived by Ralph Merkle and named after Whitfield Diffie and Martin Hellman.DH is one of the earliest practical examples of public key exchange implemented within the field of cryptography. Published in 1976 by Diffie and Hellman, this is the earliest publicly known work that proposed the idea of a private key and a corresponding public key.

Traditionally, secure encrypted communication between two parties required that they first exchange keys by some secure physical means, such as paper key lists transported by a trusted courier. The Diffie–Hellman key exchange method allows two parties that have no prior knowledge of each other to jointly establish a shared secret key over an insecure channel. This key can then be used to encrypt subsequent communications using a symmetric-key cipher.

Diffie–Hellman is used to secure a variety of Internet services. However, research published in October 2015 suggests that the parameters in use for many DH Internet applications at that time are not strong enough to prevent compromise by very well-funded attackers, such as the security services of some countries.

Code:
```
from math import sqrt

def prime_checker(n):
    prime_or_not = 0
    if n > 1:
        for i in range(2, int(sqrt(n)) + 1):
            if (n % i == 0):
                prime_or_not = 1
                break
        if (prime_or_not == 0):
            return True
        else:
            return False
    return False


def primitive_checker(p,g):
    l = []
    for i in range(0,p-1):
        temp = int(pow(g,i,p))
        if temp in l:
            return False
        else:
            l.append(temp)
    return True


def diffie_hellman():
    prime = int(input('Prime number: '))
    if not prime_checker(prime):
        prime = int(input("ERR!!! Number not prime -- Try again: "))

    primitive = int(input('Enter the generator(Primitive root of Prime number): '))
    if not primitive_checker(prime,primitive):
        primitive = int(input("ERR!!! Number should be primitive root of P\n Please Enter again: "))

    x = int(input('\nSecret x: '))
    y = int(input('Secret y: '))
    X = int(pow(primitive,x,prime))
    Y = int(pow(primitive,y,prime))
    ka = int(pow(Y,x,prime))
    kb = int(pow(X,y,prime))

    print('\nResults are as follows \nSecret key K1 :',ka)
    print('Secret Key K2 :',kb)

if __name__  == '__main__':
    diffie_hellman()
```

Screenshots:



```
Prime number:
761
Enter the generator(Primitive root of Prime number):
6

Secret x:
532
Secret y:
975

Results are as follows
Secret key K1 : 760
Secret Key K2 : 760



** Process exited - Return Code: 0 **
Press Enter to exit terminal
```

Conclusion:

1. In this experiment, I understood the need to test for big prime numbers since when I had tested out for small numbers like 11, etc the program was running properly however while trying for big numbers like 761 the program took a very long time.
2. To correct this issue the prime_checker method started to check for the condition of prime number from 2 to int(sqrt(n))+1 instead of 2 to n-1 thus reducing time complexity
3. I understood the working of Diffie Hellman and how the encrypted messages can be passed in spite of having an eavesdropper in between