# Key Exchange and Elgamal Algorithms Cryptography – Lab 5

Name: Ojas Patil Reg No: 21BAI1106

## Task

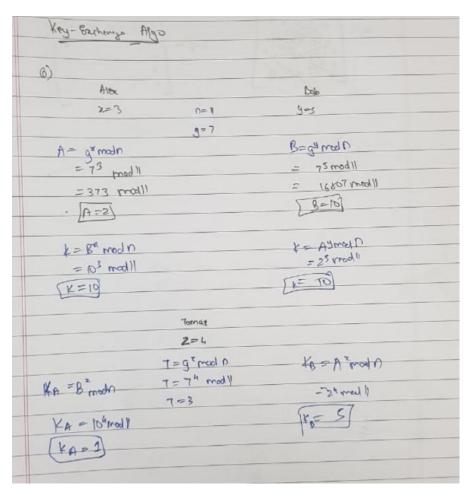
To Develop a Python-based implementation of the Key-Exchange Algorithm and Elgamal Algorithm.

## Key-Exchange Algorithm Definition

The Key Exchange algorithm revolves around the fact of mutual key derivation by the communicating entities through the mathematical process without actually physically sharing the keys.

# Key-Exchange Output Snapshot

## Key-Exchange Handwritten sum



#### Source Code

```
import random

class DHKE:
    def __init__(self,G,P):
        self.G_param = G
        self.P_param = P

    def generate_privatekey(self):
        self.pk = random.randrange(start = 1,stop = 10,step = 1)

    def generate_publickey(self):
        self.pub_key = pow(self.G_param,self.pk) % self.P_param

    def exchange_key(self,other_public):
```

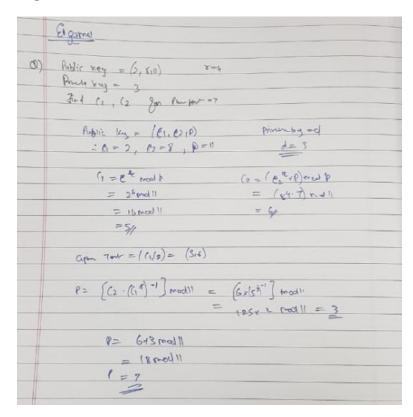
```
self.share_key = pow(other_public,self.pk) % self.P_param
#Simulating the Key Exchange b/w two entities. Let Alice and Bob be the two
entities.
Alice = DHKE(5,22)
Bob = DHKE(5,22)
Alice.generate_privatekey()
Bob.generate privatekey()
print("-----\n")
print("Alice Private Key Generated is ",Alice.pk,"\n")
print("Bob Private Key Generated is ",Bob.pk,"\n")
print("-----\n\n")
Alice.generate_publickey()
Bob.generate_publickey()
print("-----\n")
print("Alice Public Key Generated is ",Alice.pub_key,'\n')
print("Bob Public Key Generated is ",Bob.pub key,'\n')
print("-----\n\n")
#Alice & Bob Exchange each others key now.
Alice.exchange key(Bob.pub key)
Bob.exchange_key(Alice.pub_key)
print("-----\n")
print("Shared Key Generated now by Alice : ",Alice.share_key,'\n')
print("Shared Key Generated now by Bob : ",Bob.share key,'\n')
print("-----End of Shared Key Derieved-----\n")
```

# **Elgamal Algorithm Definition**

The Elgamal algorithm uses asymmetric key encryption for communicating between two parties and encrypting the message. This cryptosystem is based on the difficulty of finding discrete logarithms in a cyclic group that is even if we know ga and gk, it is extremely difficult to compute gak.

## **Elgamal Output Snapshot**

## **Elgamal Handwritten sum**



#### Source Code

```
import random
from math import pow
a = random.randint(2, 10)
def gcd(a, b):
   if a < b:
       return gcd(b, a)
   elif a % b == 0:
       return b;
   else:
        return gcd(b, a % b)
# Generating large random numbers
def gen_key(q):
    key = random.randint(pow(10, 20), q)
   while gcd(q, key) != 1:
        key = random.randint(pow(10, 20), q)
   return key
# Modular exponentiation
def power(a, b, c):
   x = 1
   y = a
   while b > 0:
        if b % 2 != 0:
           x = (x * y) \% c
       y = (y * y) % c
       b = int(b / 2)
    return x % c
# Asymmetric encryption
def encrypt(msg, q, h, g):
   en_msg = []
   k = gen_key(q)# Private key for sender
    s = power(h, k, q)
   p = power(g, k, q)
```

```
for i in range(0, len(msg)):
        en_msg.append(msg[i])
   print("g^k used : ", p)
   print("g^ak used : ", s)
   for i in range(0, len(en_msg)):
        en_msg[i] = s * ord(en_msg[i])
   return en_msg, p
def decrypt(en_msg, p, key, q):
   dr msg = []
   h = power(p, key, q)
   for i in range(0, len(en_msg)):
        dr_msg.append(chr(int(en_msg[i]/h)))
    return dr_msg
# Driver code
def main():
   msg = 'crypto ojas patil'
   print("Original Message :", msg)
   q = random.randint(pow(10, 20), pow(10, 50))
   g = random.randint(2, q)
   key = gen_key(q)# Private key for receiver
   h = power(g, key, q)
   print("g used : ", g)
   print("g^a used : ", h)
   en_msg, p = encrypt(msg, q, h, g)
   dr_msg = decrypt(en_msg, p, key, q)
   dmsg = ''.join(dr_msg)
   print("Decrypted Message :", dmsg);
if __name__ == '__main__':
   main()
```

Conclusion		
The implementation of Key Exchange Algorithm and Elgamal Algorithm in Python successfully demonstrates the core components of these algorithms and provides a foundational understanding of both the algorithms and its practical application in securing data.		