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

```
Command Prompt
Microsoft Windows [Version 10.0.19045.4780]
(c) Microsoft Corporation. All rights reserved.

C:\Users\ojasa>cd C:\VIT\sem 7\crypto lab\lab5

C:\VIT\sem 7\crypto lab\lab5>python key_exchange.py
------------Private Keys------------------

Alice Private Key Generated is  6

Bob Private Key Generated is  8

------------End of Private Keys------------


------------Public Keys------------------

Alice Public Key Generated is  5

Bob Public Key Generated is  15

------------End of Public Keys-----------


------------Shared Key Derieved------------------

Shared Key Generated now by Alice :  15

Shared Key Generated now by Bob :  15

------------End of Shared Key Derieved-----------
```

Key-Exchange Handwritten sum

Key-Exchange Algo

(6)

Alex

$x = 3$

$n = 11$

$g = 7$

Bob

$y = 5$

$A = g^x \bmod n$
$= 7^3 \bmod 11$
$= 373 \bmod 11$
$\boxed{A = 2}$

$B = g^y \bmod n$
$= 7^5 \bmod 11$
$= 16807 \bmod 11$
$\boxed{B = 10}$

$k = B^x \bmod n$
$= 10^3 \bmod 11$
$\boxed{K = 10}$

$k = A^y \bmod n$
$= 2^5 \bmod 11$
$\boxed{k = 10}$

Tomat

$z = 4$

$T = g^z \bmod n$
$T = 7^4 \bmod 11$
$T = 3$

$K_A = B^z \bmod n$
$K_A = 10^4 \bmod 11$
$\boxed{K_A = 1}$

$K_B = A^z \bmod n$
$= 2^4 \bmod 11$
$\boxed{K_B = 5}$

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("------------Private Keys------------------\n")
print("Alice Private Key Generated is ",Alice.pk,"\n")

print("Bob Private Key Generated is ",Bob.pk,"\n")
print("------------End of Private Keys------------\n\n")


Alice.generate_publickey()

Bob.generate_publickey()

print("------------Public Keys------------------\n")
print("Alice Public Key Generated is ",Alice.pub_key,'\n')

print("Bob Public Key Generated is ",Bob.pub_key,'\n')
print("------------End of Public Keys-----------\n\n")

#Alice & Bob Exchange each others key now.

Alice.exchange_key(Bob.pub_key)
Bob.exchange_key(Alice.pub_key)

print("------------Shared Key Derieved------------------\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

```
Command Prompt
Microsoft Windows [Version 10.0.19045.4780]
(c) Microsoft Corporation. All rights reserved.

C:\Users\ojasa>cd C:\VIT\sem 7\crypto lab\lab5

C:\VIT\sem 7\crypto lab\lab5>python elgamal.py
Original Message : crypto ojas patil
g used :  5469881979899784562796932762848707781852928974327
g^a used :  5598113192593237826335297409550570826235280993823
g^k used :  2108817551264003205263046584886859466879564077501l
g^ak used :  1284987839408175837229633252381749664258836870449l
Decrypted Message : crypto ojas patil

C:\VIT\sem 7\crypto lab\lab5>
```

Elgamal Handwritten sum

Elgamal

Q) Public key $= (2, 8, 11)$ $r = 4$
Private key $= 3$
Find C_1, C_2 for Plaintext $= 7$

Public key $= (e_1, e_2, p)$ Private key $= d$
$\therefore e_1 = 2,\ e_2 = 8,\ p = 11$ $d = 3$

$C_1 = e_1^r \bmod p$
$= 2^4 \bmod 11$
$= 16 \bmod 11$
$= 5$

$C_2 = (e_2^r \times P) \bmod p$
$= (8^4 \cdot 7) \bmod 11$
$= 6$

Cipher Text $= (C_1, C_2) = (5, 6)$

$P = [C_2 \cdot (C_1^d)^{-1}] \bmod 11 = [6 \times (5^3)^{-1}] \bmod 11$
$= 125 \times 2 \bmod 11 = 3$

$P = 6 \times 3 \bmod 11$
$= 18 \bmod 11$
$P = 7$

Source Code

```python
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)
```

```python
    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.