Name:-Amrit Ranjan

Reg No:-21BCE1847

Cryptography And Network Security

Assignment 8

Faculty Name:-Dr Tapabarta Roy

El Gamal Digital Signature Algorithm And El Gamal Encryption algorithm



Ques:- Construct a El Gamal Digital Signature Algorithm as well as El Gamal Encryption algorithm between two entities choosing separate sets of parameters for each of them to ensure confidentiality and authenticity .

Code For El Gamal Encryption Algorithm:-

Code:

# Python program to illustrate ElGamal encryption

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 = 'encryption'

    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)

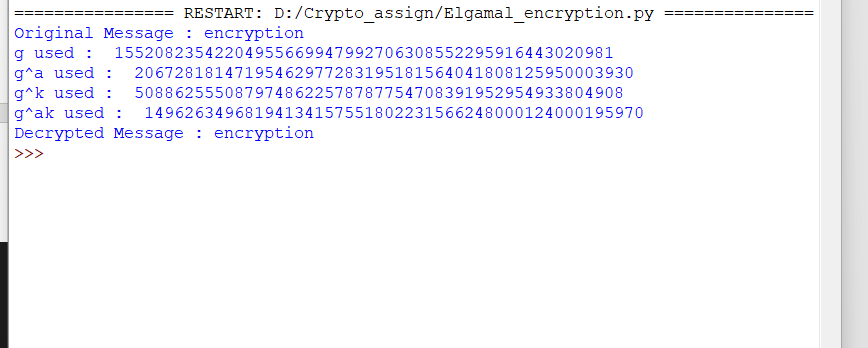
    dmsg = ''.join(dr\_msg)

    print("Decrypted Message :", dmsg);

if \_\_name\_\_ == '\_\_main\_\_':

    main()

Output:-



Code For El Gamal Digital Signature Algorithm:-

Code:-

import random

import hashlib

# Extended Euclidean Algorithm

def extended\_gcd(a, b):

    if a == 0:

        return (b, 0, 1)

    else:

        gcd, x, y = extended\_gcd(b % a, a)

        return (gcd, y - (b // a) \* x, x)

# Modular inverse using Extended Euclidean Algorithm

def mod\_inverse(a, m):

    gcd, x, y = extended\_gcd(a, m)

    if gcd != 1:

        raise Exception('Modular inverse does not exist')

    else:

        return x % m

# Fast modular exponentiation

def fast\_power(base, exp, mod):

    result = 1

    base = base % mod

    while exp > 0:

        if exp % 2 == 1:

            result = (result \* base) % mod

        exp = exp // 2

        base = (base \* base) % mod

    return result

# ElGamal key generation

def generate\_keys(p, g):

    x = random.randint(2, p - 2)

    y = fast\_power(g, x, p)

    return x, y

# ElGamal digital signature

def sign(message, p, g, x):

    # Generate k such that 1 < k < p-1 and gcd(k, p-1) = 1

    k = random.randint(2, p - 2)

    while extended\_gcd(k, p - 1)[0] != 1:

        k = random.randint(2, p - 2)

    r = fast\_power(g, k, p)

    h = int(hashlib.sha1(message.encode()).hexdigest(), 16)

    s = (mod\_inverse(k, p - 1) \* (h - x \* r)) % (p - 1)

    return (r, s)

# ElGamal signature verification

def verify(message, signature, p, g, y):

    r, s = signature

    if not (0 < r < p and 0 < s < p - 1):

        return False

    h = int(hashlib.sha1(message.encode()).hexdigest(), 16)

    left = fast\_power(y, r, p) \* fast\_power(r, s, p) % p

    right = fast\_power(g, h, p)

    return left == right

# Example usage

if \_\_name\_\_ == "\_\_main\_\_":

    # Parameters

    p = 101

    g = 2

    # Generate keys

    x, y = generate\_keys(p, g)

    # Message to be signed

    message = "Hello, World!"

    # Sign the message

    signature = sign(message, p, g, x)

    print("Signature:", signature)

    # Verify the signature

    is\_verified = verify(message, signature, p, g, y)

    if is\_verified:

        print("Signature is verified.")

    else:

        print("Signature verification failed.")

Output:-

