### Assignment no 8: Diffie-Hellman Encryption

**2020BTECS00085**

**MANJIRI CHANDURE**

**BATCH : B5**

Diffie-Hellman key exchange is a method of digital encryption that securely exchanges cryptographic keys between two parties over a public channel without their conversation being transmitted over the internet. The two parties use symmetric cryptography to encrypt and decrypt their messages. Published in 1976 by Whitfield Diffie and Martin Hellman, it was one of the first practical examples of public key cryptography.

The Diffie-Hellman algorithm is being used to establish a shared secret that can be used for secret communications while exchanging data over a public network using the elliptic curve to generate points and get the secret key using the parameters.

For the sake of simplicity and practical implementation of the algorithm, we will consider only 4 variables, one prime q and a (a primitive root of q) and two private values Xa and Xb.

p and a are both publicly available numbers. Users (say Alice and Bob) pick private values Xa and Xb and they generate a key and exchange it publicly. The opposite person receives the key and that generates a secret key, after which they have the same secret key to encrypt.

public key: yA = axA mod q

public key: yB = axB mod q,

K = yAxB mod q  (which **B** can compute)

K = yBxA mod q  (which **A** can compute)

K is same secret key.

Client 1 program:

import asyncio

import websockets

import json

import math

X = 0

def isPrime( n):

    # Corner cases

    if (n <= 1):

        return False

    if (n <= 3):

        return True

    # This is checked so that we can skip

    # middle five numbers in below loop

    if (n % 2 == 0 or n % 3 == 0):

        return False

    i = 5

    while(i \* i <= n):

        if (n % i == 0 or n % (i + 2) == 0) :

            return False

        i = i + 6

    return True

# power calculation using binary exponentiation with modulo arithmetic

def power(x, y, p) :

    res = 1     # Initialize result

    x = x % p

    if (x == 0) :

        return 0

    while (y > 0) :

        # If y is odd, multiply

        # x with result

        if ((y & 1) == 1) :

            res = (res \* x) % p

        # y must be even now

        y = y >> 1      # y = y/2

        x = (x \* x) % p

    return res

def findPrimefactors(s, n):

    # Print the number of 2s that divide n

    while (n % 2 == 0) :

        s.add(2)

        n = n // 2

    # n must be odd at this point. So we can

    # skip one element (Note i = i +2)

    for i in range(3, int(math.sqrt(n)), 2):

        # While i divides n, print i and divide n

        while (n % i == 0) :

            s.add(i)

            n = n // i

    # This condition is to handle the case

    # when n is a prime number greater than 2

    if (n > 2) :

        s.add(n)

# Function to find smallest primitive

# root of n

def findPrimitive( n) :

    s = set()

    # Check if n is prime or not

    if (isPrime(n) == False):

        return -1

    # Find value of Euler Totient function

    # of n. Since n is a prime number, the

    # value of Euler Totient function is n-1

    # as there are n-1 relatively prime numbers.

    phi = n - 1

    # Find prime factors of phi and store in a set

    findPrimefactors(s, phi)

    # Check for every number from 2 to phi

    for r in range(2, phi + 1):

        # Iterate through all prime factors of phi.

        # and check if we found a power with value 1

        flag = False

        for it in s:

            # Check if r^((phi)/primefactors)

            # mod n is 1 or not

            if (power(r, phi // it, n) == 1):

                flag = True

                break

        # If there was no power with value 1.

        if (flag == False):

            return r

    # If no primitive root found

    return -1

def getPrimitiveRoots(q):

    roots = []

    roots.append(findPrimitive(q))

    return roots

# getting list of all primitive roots for the prime for easy selection for user

def suggestRoots(q):

    roots = getPrimitiveRoots(q)

    print("These are the primitive roots for given prime (select any one)")

    print(roots)

def calculatePublicKey(X,alpha,q):

    Y = power(alpha,X,q)

    return Y

def calculateKey(Y,q):

    print(X)

    K = power(Y,X,q)

    return K

# User inputs for q,aplha and private key

def sendKey():

    print("Client Connected.....Starting the key exchange")

    print("Select The Prime Number q: ")

    q = (int)(input())

    print("Calculating the primitive roots (This may take some minutes...)")

    suggestRoots(q)

    alpha = (int)(input())

    print("Select Private Key [X] (<q): ")

    global X

    X = (int)(input())

    y = calculatePublicKey(X,alpha,q)

    print("Your Public Key[Y]: ", y)

    print("Sendint the Data to Client........")

    return {'y': y, 'alpha': alpha, 'q': q}

# sending key to other client which has the websocket server running at its port

async def keySender():

    dataToSend = sendKey()

    async with websockets.connect("ws://localhost:5000", ping\_interval=None) as websocket:

        await websocket.send(json.dumps(dataToSend))

        response = await websocket.recv()

        print(response)

        data = json.loads(response)

        Y = data["y"]

        q = data["q"]

        print(f'Received Data: q = {q} Y = {Y}')

        K = calculateKey(Y,q)

        print("Key Calculated at Original sender: ", K)

        print("DONE!!")

# receive key from other program - handled using starting websocket server

async def keyReceiver(websocket):

    async for message in websocket:

        print(message)

        data = json.loads(message)

        Y = data["y"]

        alpha = data["alpha"]

        q = data["q"]

        print(f'Received Data: alpha = {alpha}, q = {q}, Y = {Y}')

        print("Enter Your Private Key [X] (<q): ")

        global X

        X = (int)(input())

        Y\_ = calculatePublicKey(X,alpha, q)

        print("Your Public Key[Y]: ", Y\_)

        K = calculateKey(Y,q)

        print("Key Calculated: ", K)

        print("Sending Public key for verification")

        await websocket.send(json.dumps({'y': Y\_,'q': q}))

async def startServer():

    async with websockets.serve(keyReceiver,"localhost", 5000):

        await asyncio.Future()

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

    mode = 0

    print("Select The Mode: (0 for send 1 for receive) ")

    mode = (int)(input())

    if(mode == 1):

        asyncio.run(startServer())

    else:

        asyncio.run(keySender())

client 2:

import asyncio

import websockets

import json

import math

X = 0

def isPrime( n):

    # Corner cases

    if (n <= 1):

        return False

    if (n <= 3):

        return True

    # This is checked so that we can skip

    # middle five numbers in below loop

    if (n % 2 == 0 or n % 3 == 0):

        return False

    i = 5

    while(i \* i <= n):

        if (n % i == 0 or n % (i + 2) == 0) :

            return False

        i = i + 6

    return True

# power calculation using binary exponentiation with modulo arithmetic

def power(x, y, p) :

    res = 1     # Initialize result

    x = x % p

    if (x == 0) :

        return 0

    while (y > 0) :

        # If y is odd, multiply

        # x with result

        if ((y & 1) == 1) :

            res = (res \* x) % p

        # y must be even now

        y = y >> 1      # y = y/2

        x = (x \* x) % p

    return res

def findPrimefactors(s, n) :

    # Print the number of 2s that divide n

    while (n % 2 == 0) :

        s.add(2)

        n = n // 2

    # n must be odd at this point. So we can

    # skip one element (Note i = i +2)

    for i in range(3, int(math.sqrt(n)), 2):

        # While i divides n, print i and divide n

        while (n % i == 0) :

            s.add(i)

            n = n // i

    # This condition is to handle the case

    # when n is a prime number greater than 2

    if (n > 2) :

        s.add(n)

# Function to find smallest primitive

# root of n

def findPrimitive( n) :

    s = set()

    # Check if n is prime or not

    if (isPrime(n) == False):

        return -1

    # Find value of Euler Totient function

    # of n. Since n is a prime number, the

    # value of Euler Totient function is n-1

    # as there are n-1 relatively prime numbers.

    phi = n - 1

    # Find prime factors of phi and store in a set

    findPrimefactors(s, phi)

    # Check for every number from 2 to phi

    for r in range(2, phi + 1):

        # Iterate through all prime factors of phi.

        # and check if we found a power with value 1

        flag = False

        for it in s:

            # Check if r^((phi)/primefactors)

            # mod n is 1 or not

            if (power(r, phi // it, n) == 1):

                flag = True

                break

        # If there was no power with value 1.

        if (flag == False):

            return r

    # If no primitive root found

    return -1

def getPrimitiveRoots(q):

    roots = []

    roots.append(findPrimitive(q))

    return roots

# getting list of all primitive roots for the prime for easy selection for user

def suggestRoots(q):

    roots = getPrimitiveRoots(q)

    print("These are the primitive roots for given prime (select any one)")

    print(roots)

def calculatePublicKey(X,alpha,q):

    Y = power(alpha,X,q)

    return Y

def calculateKey(Y,q):

    print(X)

    K = power(Y,X,q)

    return K

# User inputs for q,aplha and private key

def sendKey():

    print("Client Connected.....Starting the key exchange")

    print("Select The Prime Number q: ")

    q = (int)(input())

    print("Calculating the primitive roots (This may take some minutes...)")

    suggestRoots(q)

    alpha = (int)(input())

    print("Select Private Key [X] (<q): ")

    global X

    X = (int)(input())

    y = calculatePublicKey(X,alpha,q)

    print("Your Public Key[Y]: ", y)

    print("Sendint the Data to Client........")

    return {'y': y, 'alpha': alpha, 'q': q}

# sending key to other client which has the websocket server running at its port

async def keySender():

    dataToSend = sendKey()

    async with websockets.connect("ws://localhost:5000", ping\_interval=None) as websocket:

        await websocket.send(json.dumps(dataToSend))

        response = await websocket.recv()

        print(response)

        data = json.loads(response)

        Y = data["y"]

        q = data["q"]

        print(f'Received Data: q = {q} Y = {Y}')

        K = calculateKey(Y,q)

        print("Key Calculated at Original sender: ", K)

        print("DONE!!")

# receive key from other program - handled using starting websocket server

async def keyReceiver(websocket):

    async for message in websocket:

        print(message)

        data = json.loads(message)

        Y = data["y"]

        alpha = data["alpha"]

        q = data["q"]

        print(f'Received Data: alpha = {alpha}, q = {q}, Y = {Y}')

        print("Enter Your Private Key [X] (<q): ")

        global X

        X = (int)(input())

        Y\_ = calculatePublicKey(X,alpha, q)

        print("Your Public Key[Y]: ", Y\_)

        K = calculateKey(Y,q)

        print("Key Calculated: ", K)

        print("Sending Public key for verification")

        await websocket.send(json.dumps({'y': Y\_,'q': q}))

async def startServer():

    async with websockets.serve(keyReceiver,"localhost", 5000):

        await asyncio.Future()

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

    mode = 0

    print("Select The Mode: (0 for send 1 for receive) ")

    mode = (int)(input())

    if(mode == 1):

        asyncio.run(startServer())

    else:

        asyncio.run(keySender())

Output:

