Roll. No. A016	Name: Varun Khadayate
Class B.Tech CsBs	Batch: 1
Date of Experiment: 10-09-2022	Subject: Cryptology

Aim

To implement Diffie Hellman Algorithm.

Theory

The purpose of the algorithm is to enable two users to securely exchange a key that can then be used for subsequent symmetric encryption of messages. The algorithm itself is limited to the exchange of secret values.

The Diffie-Hellman algorithm depends for its effectiveness on the difficulty of computing discrete logarithms. Briefly, we can define the discrete logarithm in thefollowing way. Recall from Chapter 8 that a primitive root of a prime number *p* isone whose powers modulo *p* generate all the integers from 1 to $p - 1$. That is, if is a primitive root of the prime number *p*, then the numbers

$$a \bmod p,\ a^2 \bmod p,\ \subset,\ a^{p-1} \bmod p$$

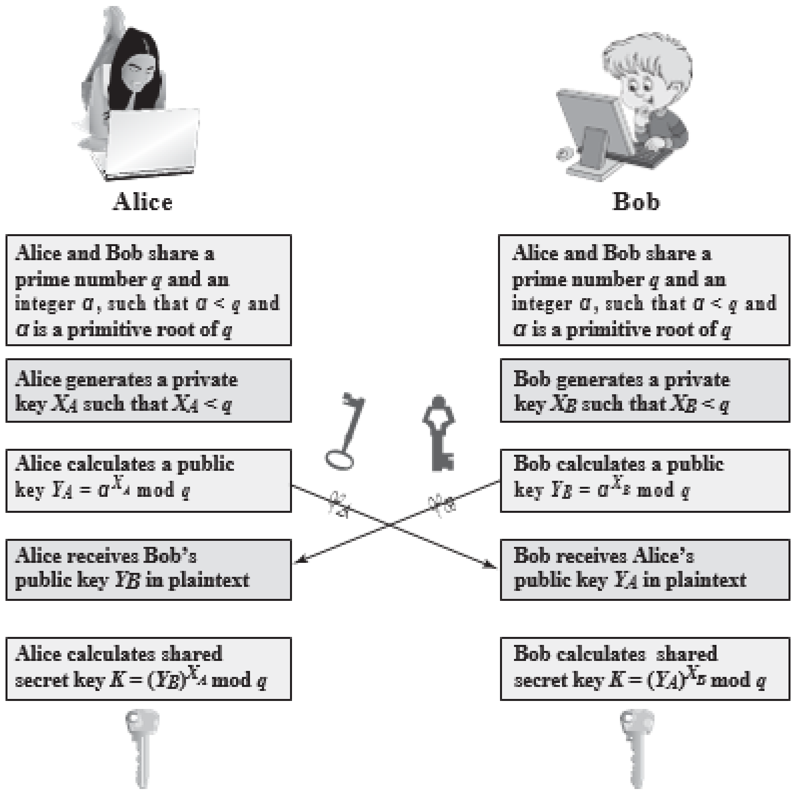
are distinct and consist of the integers from 1 through $p - 1$ in some permutation.

For any integer *b* and a primitive root of prime number *p*, we can find a unique exponent *i* such that

$$b \ K \ a^i (\bmod\ p) \qquad \text{where } 0 \ \ldots \ i \ \ldots \ (p - 1)$$

The exponent *i* is referred to as the **discrete logarithm** of *b* for the base , mod *p*. We express this value as $\text{dlog}_{a,p}(b)$. See Chapter 8 for an extended discussion of discrete logarithms.

For this scheme, there are two publicly known numbers: a prime number *q* and an integer a that is a primitive root of *q*. Suppose the users A and B wish to create a shared key.


Alice
Bob
Alice and Bob share a prime number q and an integer α, such that $\alpha < q$ and α is a primitive root of q
Alice and Bob share a prime number q and an integer α, such that $\alpha < q$ and α is a primitive root of q
Alice generates a private key X_A such that $X_A < q$
Bob generates a private key X_B such that $X_B < q$
Alice calculates a public key $Y_A = \alpha^{X_A} \bmod q$
Bob calculates a public key $Y_B = \alpha^{X_B} \bmod q$
Y_A
Y_B
Alice receives Bob's public key Y_B in plaintext
Bob receives Alice's public key Y_A in plaintext
Alice calculates shared secret key $K = (Y_B)^{X_A} \bmod q$
Bob calculates shared secret key $K = (Y_A)^{X_B} \bmod q$

User A selects a random integer X_A 6 q and computes $Y_A = a^{X_A} \bmod q$. Similarly, user B independently selects a random integer X_B 6 q and computes $Y_B = a^{X_B} \bmod q$. Each side keeps the X value private and makes the Y value avail-
able publicly to the other side. Thus, X_A is A's private key and Y_A is A's correspond- ing public key, and similarly for B. User A computes the key as $K = (Y_B)^{X_A} \bmod q$ and user B computes the key as $K = (Y_A)^{X_B} \bmod q$. These two calculations produceidentical results:

$$\begin{aligned}
K &= (Y_B)^{X_A} \bmod q \\
&= (a^{X_B} \bmod q)^{X_A} \bmod q \\
&= (a^{X_B})^{X_A} \bmod q && \text{by the rules of modular arithmetic} \\
&= a^{X_B X_A} \bmod q \\
&= (a^{X_A})^{X_B} \bmod q \\
&= (a^{X_A} \bmod q)^{X_B} \bmod q \\
&= (Y_A)^{X_B} \bmod q
\end{aligned}$$

The result is that the two sides have exchanged a secret value. Typically, thissecret value is used as shared symmetric secret key. Now consider an adversary who can observe the key exchange and wishes to determine the secret key K. Because X_A and X_B are private, an adversary only has the following ingredients to work with: q, ,Y_A, and Y_B. Thus, the adversary is forced to take a discrete logarithm to determine the key. For example, to determine the private key of user B, an adversary must compute

$$X_B = \mathrm{dlog}_{a,q}(Y_B)$$

The adversary can then calculate the key K in the same manner as user B calculatesit. That is, the adversary can calculate K as

$$K = (Y_A)^{X_B} \bmod q$$

The security of the Diffie-Hellman key exchange lies in the fact that, whileit is relatively easy to calculate exponentials modulo a prime, it is very difficult to calculate discrete logarithms. For large primes, the latter task is considered infeasible.

Here is an example. Key exchange is based on the use of the prime number $q = 353$ and a primitive root of 353, in this case a $= 3$. A and B select private keys $X_A = 97$ and $X_B = 233$, respectively. Each computes its public key:

A computes $Y_A = 3^{97} \bmod 353 = 40$.

B computes $Y_B = 3^{233} \bmod 353 = 248$.

After they exchange public keys, each can compute the common secret key:

A computes $K = (Y_B)^{X_A} \bmod 353 = 248^{97} \bmod 353 = 160$.

B computes $K = (Y_A)^{X_B} \bmod 353 = 40^{233} \bmod 353 = 160$.

We assume an attacker would have available the following information:

$$q = 353; \text{a} = 3; Y_A = 40; Y_B = 248$$

In this simple example, it would be possible by brute force to determine the secret key 160. In particular, an attacker E can determine the common key by discover- ing a solution to the equation $3^a \bmod 353 = 40$ or the equation $3^b \bmod 353 = 248$. The brute-force approach is to calculate powers of 3 modulo 353, stopping when the result equals either 40 or 248. The desired answer is reached with the exponentvalue of 97, which provides $3^{97} \bmod 353 = 40$.

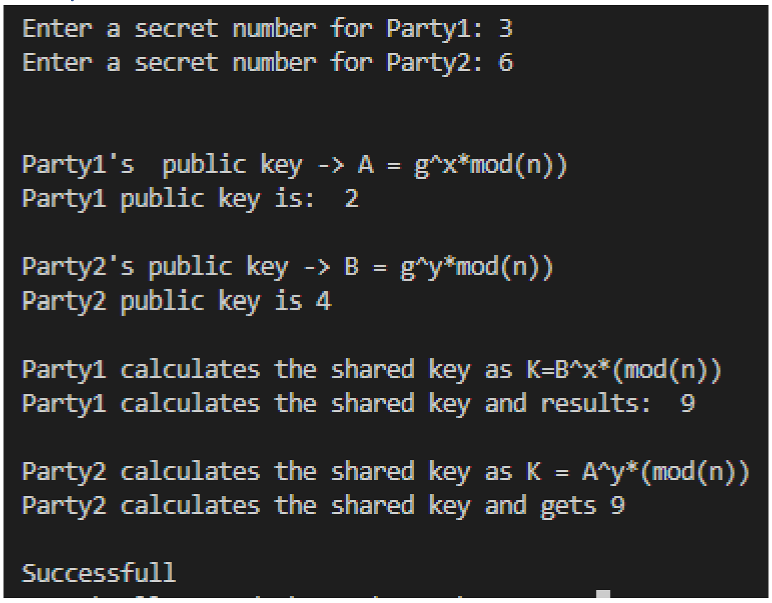
With larger numbers, the problem becomes impractical.

Code

```
n=int(input("Enter the prime number to be considered: "))
# n = 11
g=int(input("Enter the primitive root: "))
# g = 7
x=int(input("Enter a secret number for Party1: "))
# x = 3
y=int(input("Enter a secret number for Party2: "))
# y = 6
print("\n")

print ("Party1's  public key -> A = g^x*mod(n))")
alicepublic=(g**x)%n
print ("Party1 public key is: ",alicepublic, "\n")

print ("Party2's public key -> B = g^y*mod(n))")
bobpublic=(g**y)%n
print ("Party2 public key is", bobpublic, "\n")

print ("Party1 calculates the shared key as K=B^x*(mod(n))")
alicekey=(bobpublic**x)%n
print ("Party1 calculates the shared key and results: ",alicekey, "\n")

print ("Party2 calculates the shared key as K = A^y*(mod(n))")
bobkey =(alicepublic**y)%n
```

```python
print ("Party2 calculates the shared key and gets", bobkey, "\n")

if alicekey == bobkey:
    print("Successfull")
else:
    print("Un-Succesful")
```

Output



```
Enter a secret number for Party1: 3
Enter a secret number for Party2: 6


Party1's  public key -> A = g^x*mod(n))
Party1 public key is:  2

Party2's public key -> B = g^y*mod(n))
Party2 public key is 4

Party1 calculates the shared key as K=B^x*(mod(n))
Party1 calculates the shared key and results:  9

Party2 calculates the shared key as K = A^y*(mod(n))
Party2 calculates the shared key and gets 9

Successfull
```

Conclusion

Hence, we were able to perform Diffie Hellman Algorithm.