|  |  |
| --- | --- |
| 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 the following way. Recall from Chapter 8 that a primitive root of a prime number *p* is one 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* mod *p*, *a*2 mod *p*, c, *ap*-1 mod *p*

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

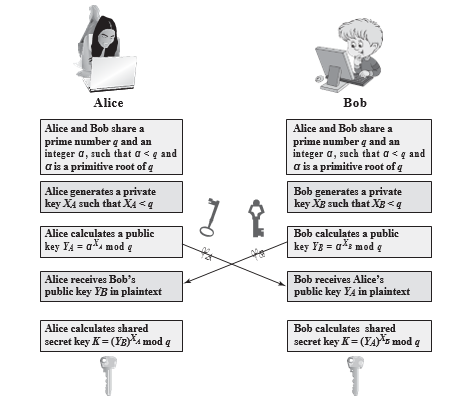
For any integer *b* and a primitive root of prime number *p*, we can find a

unique exponent *i* such that

*b* K *ai* (mod *p*) where 0 … *i* … (*p* - 1)

The exponent *i* is referred to as the **discrete logarithm** of *b* for the base , mod *p*. We express this value as 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.



User A selects a random integer *XA* 6 *q* and computes *YA* = a*XA* mod *q*. Similarly, user B independently selects a random integer *XB* 6 *q* and computes *YB* = a*XB* mod *q*. Each side keeps the *X* value private and makes the *Y* value avail-

able publicly to the other side. Thus, *XA* is A’s private key and *YA* is A’s correspond- ing public key, and similarly for B. User A computes the key as *K* = (*YB*)*XA* mod *q* and user B computes the key as *K* = (*YA*)*XB* mod *q*. These two calculations produce identical results:

*K* = (*YB*)*XA* mod *q*

= (a*XB* mod *q*)*XA* mod *q*

= (a*XB*)*XA* mod *q* by the rules of modular arithmetic

= a*XBXA* mod *q*

= (a*XA*)*XB* mod *q*

= (a*XA* mod *q*)*XB* mod *q*

= (*YA*)*XB* mod *q*

The result is that the two sides have exchanged a secret value. Typically, this secret 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 *XA* and *XB* are private, an adversary only has the following ingredients to work with: *q*, , *YA*, and *YB*. 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

*XB* = dloga,*q*(*YB*)

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

*K* = (*YA*)*XB* mod *q*

The security of the Diffie-Hellman key exchange lies in the fact that, while it 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 *XA* = 97 and *XB* = 233, respectively. Each computes its public key:

A computes *YA* = 397 mod 353 = 40.

B computes *YB* = 3233 mod 353 = 248.

After they exchange public keys, each can compute the common secret key: A computes *K* = (*YB*)*XA* mod 353 = 24897 mod 353 = 160.

B computes *K* = (*YA*)*XB* mod 353 = 40233 mod 353 = 160.

We assume an attacker would have available the following information:

*q* = 353; a = 3; *YA* = 40; *YB* = 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* mod 353 = 40 or the equation 3*b* mod 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 exponent value of 97, which provides 397 mod 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

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

if alicekey == bobkey:

    print("Successfull")

else:

    print("Un-Succesful")

# Output

Text

Description automatically generated

# Conclusion

Hence, we were able to perform Diffie Hellman Algorithm.