# Aim:

To implement Diffie Hellman Key Exchange Algorithm

**Language used: Python**

# Theory:

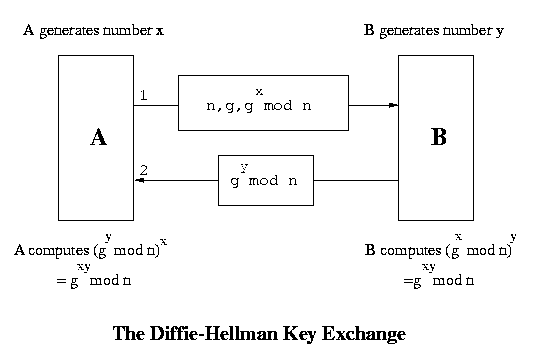
1. **Need for Key exchange algorithm**

But the problem is, letting two parties have a shared key is not easy. ... The key can't just be sent through ordinary methods because anyone who gets hold of it would then be able to decrypt all the files that the two parties would be sending to one another.

1. **Basics and Objective of Diffie-Hellman Algorithm**

Diffie-Hellman Key Exchange is an asymmetric cryptographic protocol for key exchange and its security is based on the computational hardness of solving a discrete logarithm problem. This module explains the discrete logarithm problem and describes the Diffie-Hellman Key Exchange protocol and its security issues, for example, against a man-in-the-middle attack.

1. **Block diagram of DH algorithm**



## Example of DH Algorithm

|  |
| --- |
| Suppose that two parties A and B wish to set up a common secret key (D-H key) between themselves using the Diffie Hellman key exchange technique. They agree on 7 as the modulus and 3 as the primitive root. Party A chooses 2 and party B chooses 5 as their respective secrets. Their D-H key is-   1. 3 2. 4 3. 5 4. 6     Solution-    Given-   * n = 7 * a = 3 * Private key of A = 2 * Private key of B = 5     Step-01:    Both the parties calculate the value of their public key and exchange with each other.    Public key of A  = 3private key of A mod 7  = 32 mod 7  = 2    Public key of B  = 3private key of B mod 7  = 35 mod 7  = 5  Step-02:    Both parties calculate the value of the secret key at their respective side.    Secret key obtained by A  = 5private key of A mod 7  = 52 mod 7  = 4    Secret key obtained by B  = 2private key of B mod 7  = 25 mod 7  = 4    Finally, both parties obtain the same value of the secret key.  The value of common secret key = 4.  Thus, Option (B) is correct. |

# Implementation:

import random

import hashlib

import sys

g=9

p=1001

a=random.randint(5, 10)

b=random.randint(10,20)

A = (g\*\*a) % p

B = (g\*\*b) % p

print('g: ',g,' (a shared value), n: ',p, ' (a prime number)')

print('\nAlice calculates:')

print('a (Alice random): ',a)

print('Alice value (A): ',A,' (g^a) mod p')

print('\nBob calculates:')

print('b (Bob random): ',b)

print('Bob value (B): ',B,' (g^b) mod p')

print('\nAlice calculates:')

keyA=(B\*\*a) % p

print('Key: ',keyA,' (B^a) mod p')

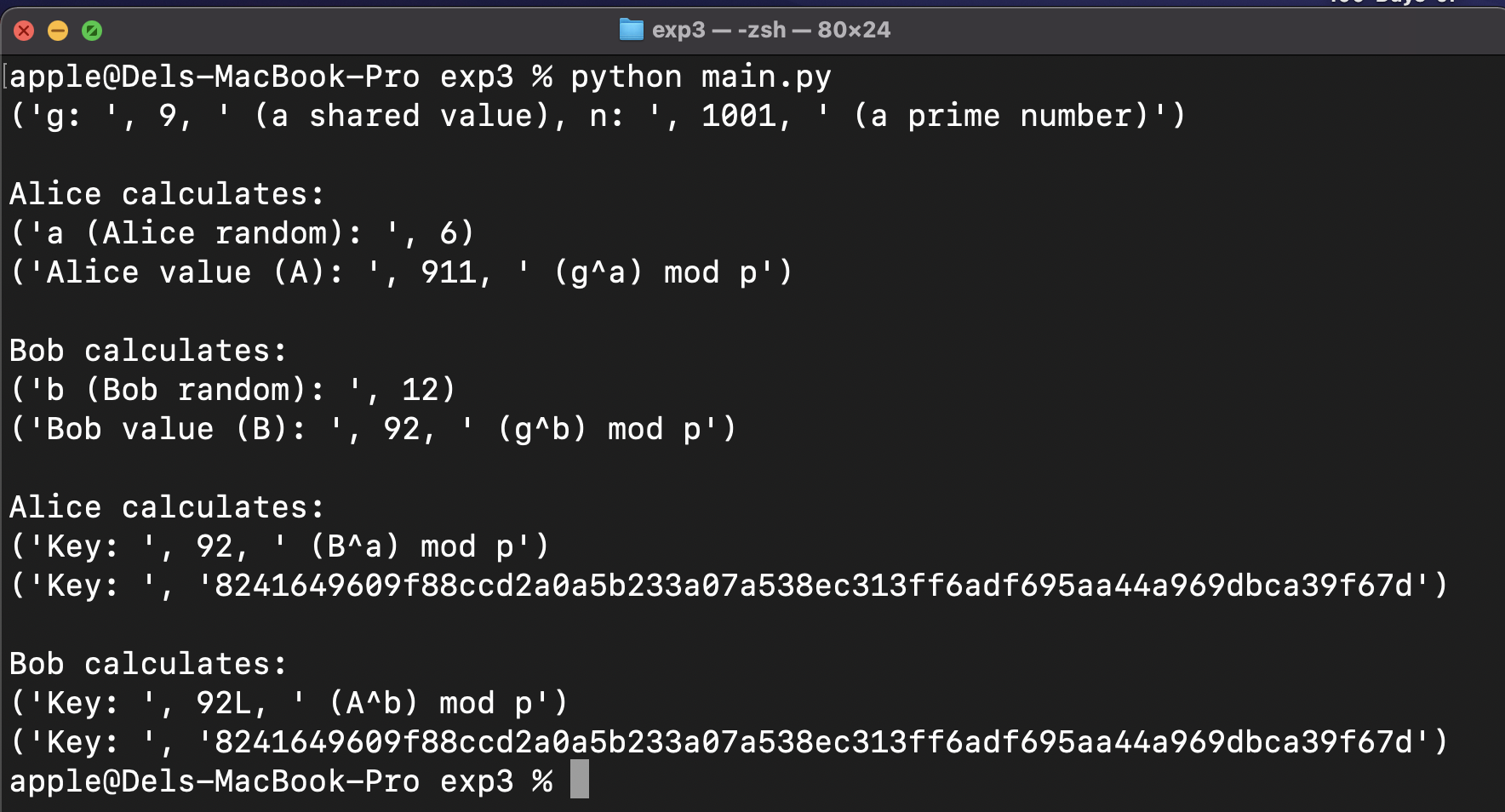
print('Key: ',hashlib.sha256(str(keyA).encode()).hexdigest())

print('\nBob calculates:')

keyB=(A\*\*b) % p

print('Key: ',keyB,' (A^b) mod p')

print('Key: ',hashlib.sha256(str(keyB).encode()).hexdigest()



# Conclusion:

In this experiment, we implemented the Diffie Hellman Key Exchange Algorithm using Python code.

# Viva Questions:

1. **What is primitive root of a number**

In modular arithmetic, a branch of number theory, a number g is a primitive root modulo n if every number a coprime to n is congruent to a power of g modulo n. That is, g is a primitive root modulo n, if for every integer a coprime to n, there is some integer k for which gk ≡ a (mod n).

1. **How to calculate primitive roots. Give example**

Numbers that have a primitive root are

1, 2, 3, 4, 5, 6, 7, 9, 10, 11, 13, 14, 17, 18, 19, 22, 23, 25, 26, 27, 29, 31, 34, 37, 38, 41, 43, 46, 47, 49, 50, 53, 54, 58, 59, 61, 62, 67, 71, 73, 74, 79, 81, 82, 83, 86, 89, 94, 97, 98, 101, 103, 106, 107, 109, 113, 118, 121, 122, 125, 127, 131, 134, 137, 139, 142, 146, 149, ...

This is Gauss's table of the primitive roots from the *Disquisitiones*. Unlike most modern authors he did not always choose the smallest primitive root. Instead, he chose 10 if it is a primitive root; if it isn't, he chose whichever root gives 10 the smallest index, and, if there is more than one, chose the smallest of them. This is not only to make hand calculation easier, but is used in § VI where the periodic decimal expansions of rational numbers are investigated.

The rows of the table are labelled with the prime powers (excepting 2, 4, and 8) less than 100; the second column is a primitive root modulo that number. The columns are labelled with the primes less than 100. The entry in row *p*, column *q* is the index of *q* modulo *p* for the given root.

For example, in row 11, 2 is given as the primitive root, and in column 5 the entry is 4. This means that 24 = 16 ≡ 5 (mod 11).

For the index of a composite number, add the indices of its prime factors.

For example, in row 11, the index of 6 is the sum of the indices for 2 and 3: 21 + 8 = 512 ≡ 6 (mod 11). The index of 25 is twice the index 5: 28 = 256 ≡ 25 (mod 11). (Of course, since 25 ≡ 3 (mod 11), the entry for 3 is 8).

1. **Is Diffie-Hellman algorithm prone to any attack:**

in diffie-hellman key exchange algorithm vulnerabilities is good defined by RSA lab :

"The Diffie-Hellman key exchange is vulnerable to a man-in-the-middle attack. In this attack, an opponent Carol intercepts Alice's public value and sends her own public value to Bob. When Bob transmits his public value, Carol substitutes it with her own and sends it to Alice. Carol and Alice thus agree on one shared key and Carol and Bob agree on another shared key. After this exchange, Carol simply decrypts any messages sent out by Alice or Bob, and then reads and possibly modifies them before re-encrypting with the appropriate key and transmitting them to the other party. This vulnerability is present because Diffie-Hellman key exchange does not authenticate the participants. Possible solutions include the use of digital signatures and other protocol variants."

and some simple example of mitm attack :

Alice "Hi Bob, it's Alice. Give me your key." → Mallory Bob

Alice Mallory "Hi Bob, it's Alice. Give me your key." → Bob

Alice Mallory ← [Bob's key] Bob

Alice ← [Mallory's key] Mallory Bob

Alice "Meet me at the bus stop!" [encrypted with Mallory's key] → Mallory Bob

Alice Mallory "Meet me in the windowless van on 22nd Ave!" [encrypted with Bob's key] → Bob