Diffie Helman Key Exchange

Prashanth.S 19MID0020

Importing the Necessary Libraries

In [1]:
```
import numpy as np
import random
import warnings
warnings.filterwarnings("ignore")
```

In [2]:
```
'''
Xa = 3 ## private key of Agent-X
Ya = 7 ## private key of Agent-Y
A = ## public key of Agent-X (shared to Agent-Y)
B = ## public key of Agent-Y (shared to Agent-X)
S = ## shared key
g = generator (unique values after the finding the primitive root of the prime modulo)
'''
```

Out[2]:
```
'\nXa = 3 ## private key of Agent-X\nYa = 7 ## private key of Agent-Y\nA = ## public key of Agent-X (shared to Agent-Y)\nB = ## public key of Agent-Y (shared to Agent-X)\nS = ## shared key \ng = generator (unique values after the finding the primitive root of the prime modulo)\n'
```

Primitive Roots Creation

In [3]:
```
def primitive_roots_table_creation(m):
    list1 = []
    b = 1
    for i in range(1,m-1):
        temp = []
        b+=1
        for j in range(1,m):
            temp.append(np.power(b,j) % (m))
        list1.append(temp)
    return list1
```

In [4]:
```
def primitive_roots_value(primitive_roots_table, m):

    # np.unique() --> returns the number bo unique values
    primitive_roots = []

    for i in primitive_roots_table:
        if (len(np.unique(i)) == m-1):
            primitive_roots.append(i[0])
    return primitive_roots
```

In [5]:
```
def public_key_generation(generator, private_key, primitive_root):
    return ((generator**private_key) % primitive_root)

def sharing(pub1, pub2):
    return (pub2, pub1)

def share_secret_key(shared_key, private_key, primitive_root):
    return ((shared_key**private_key) % primitive_root)
```

In [6]:
```
def isPrime(num):
    cnt = 0

    for i in range(2, np.int(np.sqrt(num))):

        ## composite number
        if ((num%i) == 0):
            cnt = 1
            return False

        ## prime number
        if (cnt==0):
            return True
```

Sharing the Secret Key

In [7]:
```
def start():

    ## Alice portion
    Xa = 3
    Ya = public_key_generation(generator, Xa, prime_number)
    print("Alice's public key : ",Ya)

    ## Bob portion
    Xb = 7
    Yb = public_key_generation(generator, Xb, prime_number)

    Ya, Yb = sharing(Yb, Ya)
    print("\nAfter sharing")
    print("Alice's public key : ",Ya)
    print("Bob's public key : ",Yb)

    alice_K = share_secret_key(Yb, Xa, prime_number)
    print("\nAlice's shared key : ",alice_K)

    bob_K = share_secret_key(Ya, Xb, prime_number)
    print("Bob's shared key : ",bob_K)

    if (alice_K == bob_K): return alice_K
    else: return 0
```

Selection of Primitive Root

In [8]:
```
primitive_roots_table = primitive_roots_table_creation(13)

primitive_roots = primitive_roots_value(primitive_roots_table,13)
primitive_roots
```

Out[8]:
```
[2, 6, 7, 11]
```

In [9]:
```
primitive_roots_table
```

Out[9]:
```
[[2, 4, 8, 3, 6, 12, 11, 9, 5, 10, 7, 1],
 [3, 9, 1, 3, 9, 1, 3, 9, 1, 3, 9, 1],
 [4, 3, 12, 9, 10, 1, 4, 3, 12, 9, 10, 1],
 [5, 12, 8, 1, 5, 12, 8, 1, 5, 12, 8, 1],
 [6, 10, 8, 9, 2, 12, 7, 3, 5, 4, 11, 1],
 [7, 10, 5, 9, 11, 12, 6, 3, 8, 4, 2, 1],
 [8, 12, 5, 1, 8, 12, 5, 1, 8, 12, 5, 1],
 [9, 3, 1, 9, 3, 1, 9, 3, 1, 9, 3, 1],
 [10, 9, 12, 3, 4, 1, 10, 9, 12, 3, 4, 1],
 [11, 4, 5, 3, 7, 12, 2, 9, 8, 10, 6, 1],
 [12, 1, 12, 1, 12, 1, 12, 1, 12, 1, 12, 1]]
```

In [10]:
```
m = 13
primitive_roots = []
for i in primitive_roots_table:
    if (len(np.unique(i)) == m-1):
        primitive_roots.append(i[0])
```

In [11]:
```
primitive_roots
## For a = b^i mod m where b is the primitive root for the prime modulo m
## For unique value of i --> b should be the primitive root of the prime modulo m.
## For unique value of i (generator) --> (2 or 6 or 7 or11) is the primitive root of the prime modulo 13.
```

Out[11]:
```
[2, 6, 7, 11]
```

Actual Program Starts

In [12]:
```
## So we came to know that the primitive roots
prime_number = 13

## taking a random value from the generator
if isPrime(prime_number):
    generator = random.choice(primitive_roots)

if (generator < prime_number):
    print("Continue")
    key_match = start() ## key_match will have the 'shared secret key' , if there is a primitive root for the p

else:
    print("Discontinue")
```

```
Continue
Alice's public key :  5

After sharing
Alice's public key :  5
Bob's public key :  2

Alice's shared key :  8
Bob's shared key :  8
```

In [13]:
```
## After the secret keys are matched
```

In [14]:
```
def shift_characters(str1, n):
    return ''.join(chr((ord(char) - 97 - n) % 26 + 97) for char in str1)
```

Encryption

In [15]:
```
def encryption(plain_text):
    shared_secret_key = key_match
    return shift_characters(plain_text, shared_secret_key)
```

Decryption

In [16]:
```
def decryption(cipher_text):
    shared_secret_key = -key_match
    return shift_characters(cipher_text, shared_secret_key)
```

In [17]:
```
plain_text = "prashanth"
cipher_text = encryption(plain_text)
print(cipher_text)
```

```
hjskzsflz
```

In [18]:
```
decrypt_text = decryption(cipher_text)
print(decrypt_text)
```

```
prashanth
```