Rasel, Mostafa

17-33635-1

Sec: A

Github:

**Assignment 4**

#include<stdio.h>

#include<math.h>

// Power function to return value of a ^ b mod P

long long int power(long long int a, long long int b,

long long int P)

{

if (b == 1)

return a;

else

return (((long long int)pow(a, b)) % P);

}

//Driver program

int main()

{

long long int P, G, x, a, y, b, ka, kb;

// Both the persons will be agreed upon the

// public keys G and P

P = 21; // A prime number P is taken

printf("The value of P : %lld\n", P);

G = 7; // A primitve root for P, G is taken

printf("The value of G : %lld\n\n", G);

// Alice will choose the private key a

a = 4; // a is the chosen private key

printf("The private key a for Alice : %lld\n", a);

x = power(G, a, P); // gets the generated key

// Bob will choose the private key b

b = 3; // b is the chosen private key

printf("The private key b for Bob : %lld\n\n", b);

y = power(G, b, P); // gets the generated key

// Generating the secret key after the exchange

// of keys

ka = power(y, a, P); // Secret key for Alice

kb = power(x, b, P); // Secret key for Bob

printf("Secret key for the Alice is : %lld\n", ka);

printf("Secret Key for the Bob is : %lld\n", kb);

return 0;

}

Assignment 1

#include<bits/stdc++.h>

using namespace std;

// Returns true if n is prime

bool isPrime(int 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 || n%3 == 0) return false;

for (int i=5; i\*i<=n; i=i+6)

if (n%i == 0 || n%(i+2) == 0)

return false;

return true;

}

/\* Iterative Function to calculate (x^n)%p in

O(logy) \*/

int power(int x, unsigned int y, int p)

{

int res = 1; // Initialize result

x = x % p; // Update x if it is more than or

// equal to p

while (y > 0)

{

// If y is odd, multiply x with result

if (y & 1)

res = (res\*x) % p;

// y must be even now

y = y >> 1; // y = y/2

x = (x\*x) % p;

}

return res;

}

// Utility function to store prime factors of a number

void findPrimefactors(unordered\_set<int> &s, int n)

{

// Print the number of 2s that divide n

while (n%2 == 0)

{

s.insert(2);

n = n/2;

}

// n must be odd at this point. So we can skip

// one element (Note i = i +2)

for (int i = 3; i <= sqrt(n); i = i+2)

{

// While i divides n, print i and divide n

while (n%i == 0)

{

s.insert(i);

n = n/i;

}

}

// This condition is to handle the case when

// n is a prime number greater than 2

if (n > 2)

s.insert(n);

}

// Function to find smallest primitive root of n

int findPrimitive(int n)

{

unordered\_set<int> s;

// 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.

int 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 (int r=2; r<=phi; r++)

{

// Iterate through all prime factors of phi.

// and check if we found a power with value 1

bool flag = false;

for (auto it = s.begin(); it != s.end(); it++)

{

// 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;

}

// Driver code

int main()

{

int n = 97;

cout << " Smallest primitive root of " << n

<< " is " << findPrimitive(n);

return 0;

}