Program – 10

AIM: To implement a program to calculate and verify DSA given in DSS.

Introduction and Theory

The Digital Signature Algorithm (DSA) is a Federal Information Processing Standard for digital signatures, based on the mathematical concept of modular exponentiations and the discrete logarithm problem.
The DSA algorithm works in the framework of public-key cryptosystems and is based on the algebraic properties of the modular exponentiations, together with the discrete logarithm problem (which is considered to be computationally intractable). Messages are signed by the signer's private key and the signatures are verified by the signer's corresponding public key. The digital signature provides message authentication, integrity and non-repudiation.

The first part of the DSA algorithm is the public key and private key generation, which can be described as:

- Choose a prime number q, which is called the prime divisor.
- Choose another primer number p, such that p-1 mod q = 0. p is called the prime modulus.
- Choose an integer g, such that 1 < g < p, g**q mod p = 1 and g = h**((p–1)/q) mod p. q is also called g's multiplicative order modulo p.
- Choose an integer, such that 0 < x < q.
- Compute y as g**x mod p.
- Package the public key as {p,q,g,y}.
- Package the private key as {p,q,g,x}.

The second part of the DSA algorithm is the signature generation and signature verification, which can be described as:

To generate a message signature, the sender can follow these steps:

- Generate the message digest h, using a hash algorithm like SHA1.
- Generate a random number k, such that 0 < k < q.
- Compute r as (g**k mod p) mod q. If r = 0, select a different k.
- Compute i, such that k*i mod q = 1. i is called the modular multiplicative inverse of k modulo q.
- Compute s = i*(h+r*x) mod q. If s = 0, select a different k.
- Package the digital signature as {r,s}.

Program – 10

To verify a message signature, the receiver of the message and the digital signature can follow these steps:

- Generate the message digest h, using the same hash algorithm.
- Compute w, such that s*w mod q = 1. w is called the modular multiplicative inverse of s modulo q.
- Compute u1 = h*w mod q.
- Compute u2 = r*w mod q.
- Compute v = (((g**u1)*(y**u2)) mod p) mod q.
- If v == r, the digital signature is valid.

Code

```
// #include<iostream>
// #include<string>
// #include<utility>
// #include<cmath>
#include<bits/stdc++.h>

using namespace std;

long long int power(long long int x,  long long int y, long long int
p)
{
    long long 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;
}


long long Hash(string S, int p)
{
    long long hash = 1;
    int prev = 1;
    for (int i = 0; i < S.length(); i++)
    {
        hash = (hash + int(S[i])*prev % p);
        prev = int(S[i]);
    }
    return hash;
}
```

Program – 10

```

long long int modInverse(long long int a, long long int m)
{
    long long int m0 = m;
    long long int y = 0, x = 1;

    if (m == 1)
      return 0;

    while (a > 1)
    {
        // q is quotient
        long long int q = a / m;
        long long int t = m;

        // m is remainder now, process same as
        // Euclid's algo
        m = a % m, a = t;
        t = y;

        // Update y and x
        y = x - q * y;
        x = t;
    }

    // Make x positive
    if (x < 0)
       x += m0;

    return x;
}

class DSA
{
private:
    long long int x;
    long long int g;
    long long int y;
    long long int p;
    long long int q;
public:
    DSA(int , int );
    pair<long long int, long long int> sign(string);
    bool verify(string, pair<long long int, long long int>);

};

DSA::DSA(int P, int Q)
{
    x = rand() % 100 + 1;
    p = P;
    q = Q;
    g = power(2, (p-1)/q, p);
    y = power(g, x, p);
}

pair<long long int, long long int> DSA::sign(string s)
```

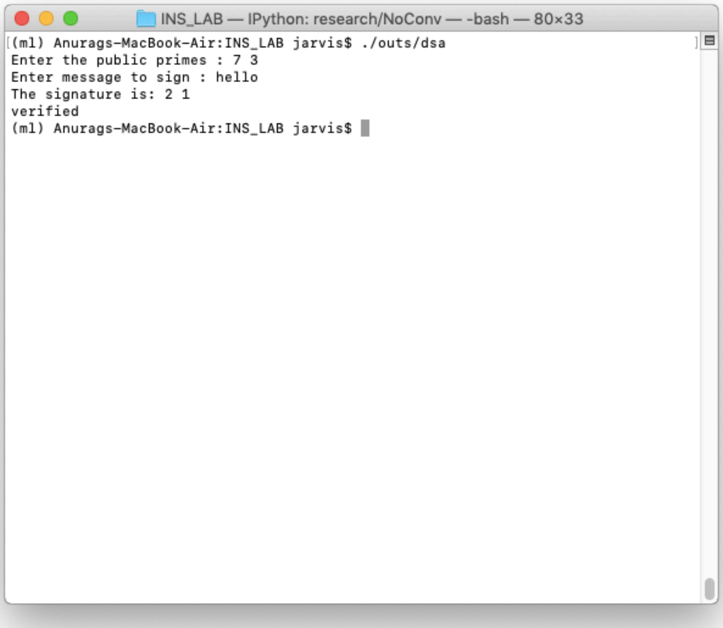
Program – 10

```
{
    long long int s1 = 0;
    long long int s2 = 0;
    long long int r;
    do
    {
        r = rand() % q + 1;
        s1 = power(g, r, p) % q;
        long long k = modInverse(r, q);
        s2 = (k * (Hash(s, p) + x * s1)) % q;
    }while(s1 == 0 || s2 == 0);
    return make_pair(s1, s2);
}
bool DSA::verify(string s, pair<long long int, long long int> sign)
{
    long long int h = Hash(s, p);
    long long int w, u1, u2, v;
    w = modInverse(sign.second, p);
    u1 = (h * w);
    // cout << u1 << endl;
    u2 = (sign.first * w);
    // cout << u2 << endl;
    v = ((power(g, u1, p) * power(y, u2, p))%p)%q;
    // cout << v << endl;
    return (abs(v) == sign.first);
}

int main()
{
    long long int p, q;
    cout << "Enter the public primes : ";
    cin >> p >> q;
    cout << "Enter message to sign : ";
    string s;
    cin >> s;
    DSA D(p, q);
    pair<long long int, long long int> sin = D.sign(s);
    cout << "The signature is: " << sin.first << ' ' << sin.second
<< endl;
    if (D.verify(s, sin))
        cout << "verified" << endl;
    else
        cout <<"Rejected" <<endl;
}
```

Program – 10

Results and Outputs:



Findings and Learnings:

1. We have implemented DSA and verified it.