SIDDAGANGA INSTITUTE OF TECHNOLOGY, TUMKUR-572103

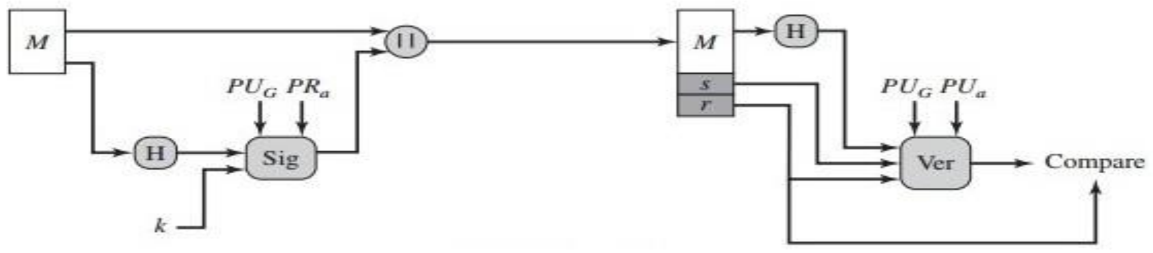
DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING CRYPTOGRAPHY AND NETWORK SECURITY LAB (7RCSL01)

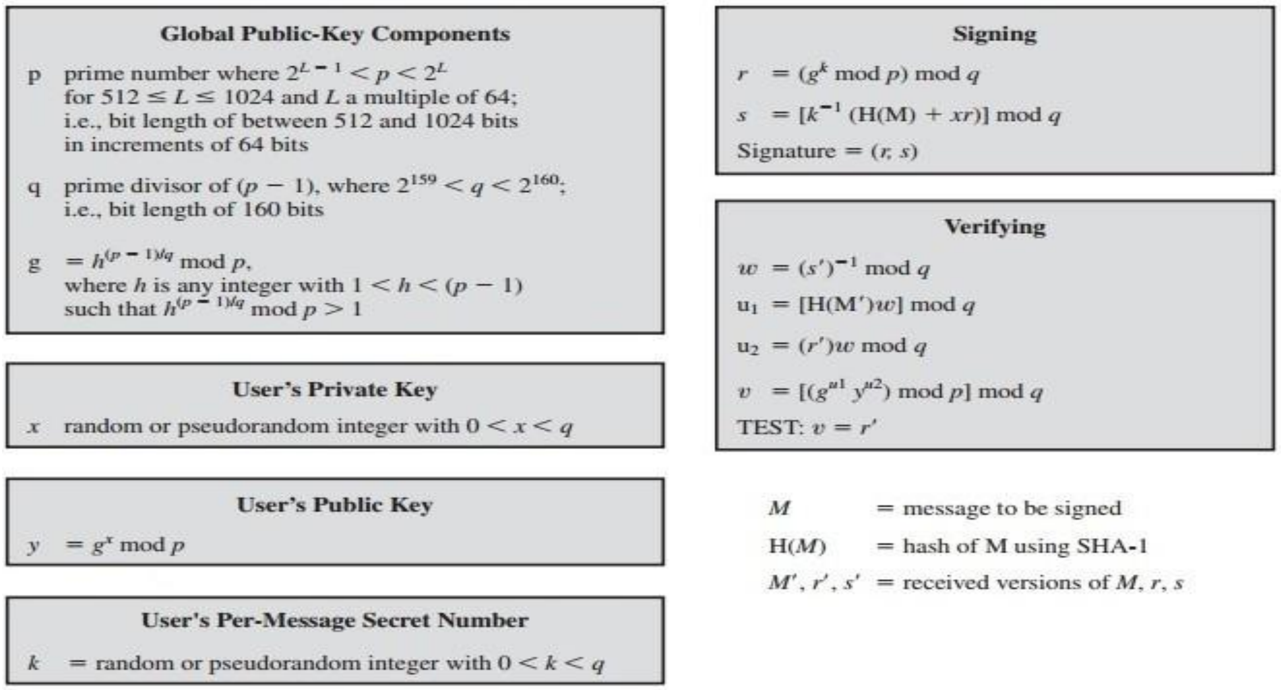
Student Name: RITI	**USN: 1SI19CS144**	**Batch No:B4**	**Date:**	
Evaluation:				
Write Up (10 marks)	**Clarity in concepts (10 marks)**	**Implementation and execution of the algorithms (10 marks)**	**Viva (05 marks)**	**Total (35 marks)**

Sl.No	Name of the Faculty In-Charge	Signature
1.	**Dr.H K Vedamurthy**	
2.	**Dr. A H Shanthakumara**	

Question No: 14. Implement DSS algorithm for signing and verification ofmessages between two parties (obtain H(M) using simple XOR method of hash computation on M).

Algorithm:





Global Public-Key Components

p prime number where $2^{L-1} < p < 2^L$ for $512 \le L \le 1024$ and L a multiple of 64; i.e., bit length of between 512 and 1024 bits in increments of 64 bits

q prime divisor of $(p - 1)$, where $2^{159} < q < 2^{160}$; i.e., bit length of 160 bits

g $= h^{(p-1)/q} \bmod p$, where h is any integer with $1 < h < (p - 1)$ such that $h^{(p-1)/q} \bmod p > 1$

User's Private Key

x random or pseudorandom integer with $0 < x < q$

User's Public Key

$y = g^x \bmod p$

User's Per-Message Secret Number

k = random or pseudorandom integer with $0 < k < q$

Signing

$r = (g^k \bmod p) \bmod q$

$s = [k^{-1}(H(M) + xr)] \bmod q$

Signature = (r, s)

Verifying

$w = (s')^{-1} \bmod q$

$u_1 = [H(M')w] \bmod q$

$u_2 = (r')w \bmod q$

$v = [(g^{u1} y^{u2}) \bmod p] \bmod q$

TEST: $v = r'$

M	= message to be signed
$H(M)$	= hash of M using SHA-1
M', r', s'	= received versions of M, r, s

CODE:-

```
# include <bits/stdc++.h>
# include <arpa/inet.h>
using namespace std;

int createServer(int port)  // TCP connection
{
   int sersock = socket(AF_INET, SOCK_STREAM, 0);
   struct sockaddr_in addr = {AF_INET, htons(port), INADDR_ANY};

   bind(sersock, (struct sockaddr *) &addr, sizeof(addr));
   cout << "\nServer Online. Waiting for client...." << endl;

   listen(sersock, 5);
   int sock = accept(sersock, NULL, NULL);
   cout << "Connection Established." << endl;
   return sock;
}

long randInRange(long low, long high) // excluding high and low
{
   return rand()%(high-(low+1)) + (low+1) ;
}

long mod(long a, long b)
{
       return a >= 0 ? (a%b) : b-(abs(a)%b) ;
}

long powermod(long a, long b, long  c)
{
   long res=1;
   for(int i=0; i<b; i++)
   {
      res = (res * a) % c;
   }
   return res;
}

long findInverse(long R , long D)
```

```
{
   int i = 0;
   long N = D; // copy D to N for taking mod
   long p[100] = {0,1};
   long q[100] = {0} ;

   while(R!=0)
   {
      q[i] = D/R ;
      long oldD = D ;
      D = R ;
      R = oldD%R ;
      if(i>1)
      {
         p[i] = mod(p[i-2] - p[i-1]*q[i-2], N) ;
      }
      i++ ;
   }
   if (i == 1) return 1;
   else       return p[i] = mod(p[i-2] - p[i-1]*q[i-2], N) ;
}

long H(long M) // Hash Function
{
       return (M ^ 1234); //hash key = 1234
}

int main()
{
   int port;  cout << "\nEnter port : "; cin >> port;
   int sock = createServer(port);

   long p, q; // prime numbers
   long r, s; // signature
   long k, x, y, g; // keys
   long M, hashval; // Message and Hash
   srand(time(NULL));

   cout << "\nEnter a large prime number, p : ";   cin >> p;
   cout << "Enter a prime number, q (p-1 divisible by q & q>2) : "; cin >> q;
```

```
    if( (p-1)%q != 0 || q <3) { cout << "\nInvalid input\n"; exit(-1); }

    cout<<"Enter message, M = "; cin >> M;

    hashval = H(M);
    cout << "\nH(M) = " << hashval << endl;

    long h;
    do{
      h = randInRange(1, p-1);       // 1 < h < p-1
      g = powermod(h,(p-1)/q, p);       //g > 1
    } while(g<=1);
    cout << "g    = " << g;

    x = randInRange(1, q);  cout << "\nServer's Private key, x = " << x;
    y = powermod(g, x, p);  cout << "\nServer's Public  key, y = " << y;
    k = randInRange(1, q);  cout << "\nSecret key, k = " << k << endl;

    //Signing
    r = powermod(g, k, p) % q;
    s = (findInverse(k,q) * (hashval + x*r )) % q;
    cout << "\nServer's Signature {r,s} = {" << r << ", " << s << "}" << endl;

    send(sock, &p, sizeof(p), 0);
    send(sock, &q, sizeof(q), 0);
    send(sock, &g, sizeof(g), 0);
    send(sock, &y, sizeof(y), 0);
    send(sock, &M , sizeof(M), 0);
    send(sock, &r, sizeof(r), 0);
    send(sock, &s, sizeof(s), 0);

    cout << "\nSent p, q, g, and public key to client.";
    cout <<"\nSent message along with signature to client." << endl << endl;
  }
```

Client program:

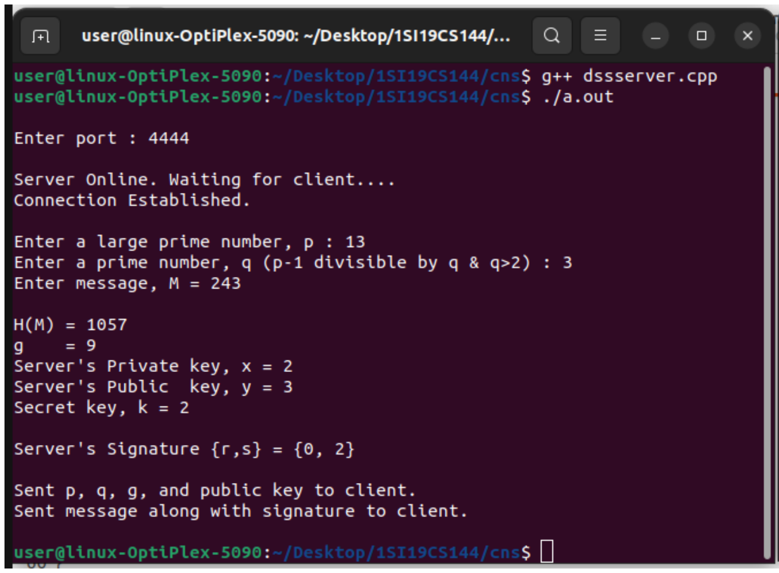
```
# include <bits/stdc++.h>
# include <arpa/inet.h>
using namespace std;

int connectToServer(const char* ip, int port)
```

```
{
   int sock = socket(AF_INET, SOCK_STREAM, 0);
   struct sockaddr_in addr = {AF_INET, htons(port),inet_addr(ip)};

   if(connect(sock, (struct sockaddr *) &addr, sizeof(addr)) < 0 ){
      cout << "\nRun server program first." << endl; exit(0);
   }else{
      cout << "\nClient is connected to Server." << endl;
   }
   return sock;
}

long mod(long a, long b)
{
   return a >= 0 ? (a%b) : b-(abs(a)%b) ;
}

long powermod(long a, long b, long  c)
{
   long res=1;
   for(int i=0; i<b; i++)
   {
      res = (res * a) % c;
   }
   return res;
}

long findInverse(long R , long D)
{
   int i = 0;
   long N = D; // copy D to N for taking mod
   long p[100] = {0,1};
   long q[100] = {0} ;

   while(R!=0)
   {
      q[i] = D/R ;
      long oldD = D ;
      D = R ;
      R = oldD%R ;
      if(i>1)
      {
         p[i] = mod(p[i-2] - p[i-1]*q[i-2], N) ;
      }
      i++ ;
   }
   if (i == 1) return 1;
   else       return p[i] = mod(p[i-2] - p[i-1]*q[i-2], N) ;
}
```

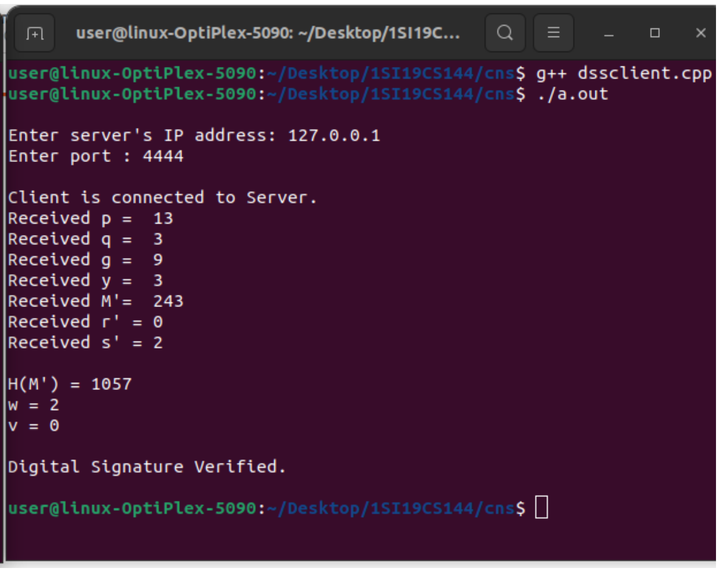
```
long H(long M)
{
   return (M ^ 1234); //hash key = 1234
}

int main()
{
   char ip[50]; cout << "\nEnter server's IP address: "; cin >> ip;
   int port;    cout << "Enter port : "; cin >> port;
   int sock = connectToServer(ip, port);

   long p, q; // prime numbers
   long r, s; // signature
   long g, y; // keys
   long M, hashval; // Message and Hash
   long w, v; // verify
   srand(time(NULL));

   recv(sock, &p, sizeof(p), 0);
   recv(sock, &q, sizeof(q), 0);
   recv(sock, &g, sizeof(g), 0);
   recv(sock, &y, sizeof(y), 0);
   recv(sock, &M , sizeof(M), 0);
   recv(sock, &r, sizeof(r), 0);
   recv(sock, &s, sizeof(s), 0);

   cout << "Received p =  " << p << endl;
   cout << "Received q =  " << q << endl;
   cout << "Received g =  " << g << endl;
   cout << "Received y =  " << y << endl;
   cout << "Received M'=  " << M << endl;
   cout << "Received r' = " << r << endl;
   cout << "Received s' = " << s << endl;

   hashval = H(M) ;
   cout << "\nH(M') = " << hashval << endl;

   //Verifying
   w = findInverse(s,q) % q;  cout << "w = " << w << endl;
   long u1 = (hashval * w) % q;
   long u2 = (r * w) % q;
   v = ((powermod(g,u1,p)*powermod(y,u2,p)) %p) %q;  cout<<"v = "<<v<<endl;
   if(v == r) cout<<"\nDigital Signature Verified. " << endl << endl;
   else     cout<<"\nDigital Signature is invalid !!!" << endl << endl;
}
```

```
        int sock=connectToServer(ip,port);

        long p,q;
        long   r,s;
        long g,y;
        long M, hashval;
        long w,v;
        srand(time(NULL));

        recv(sock, &p, sizeof(p), 0);
        recv(sock, &q, sizeof(q), 0);
        recv(sock, &g, sizeof(g), 0);
        recv(sock, &y, sizeof(y), 0);
        recv(sock, &M, sizeof(M), 0);
        recv(sock, &r, sizeof(r), 0);
        recv(sock, &s, sizeof(s), 0);

        cout << "Received p = " << p << endl;
        cout << "Received q = " << q << endl;
        cout << "Received g = " << g << endl;
        cout << "Received y = " << y << endl;
        cout << "Received M'= " << M << endl;
        cout << "Received r' = " << r << endl;
        cout << "Received s' = " << s << endl;

        hashval=H(M);
        cout<<"\nH(M')= "<<hashval<<endl;

        w = findInverse(s,q) % q;
        cout << "w = " << w << endl;
        long u1 = (hashval * w) % q;
        long u2 = (r * w) % q;
        v = ((powermod(g,u1,p)*powermod(y,u2,p)) %p) %q;
        cout<<"v = "<<v<<endl;
        if(v == r)
                cout<<"\nDigital Signature Verified. " << endl << endl;
        else
                cout<<"\nDigital Signature is invalid !!!" << endl << endl;
}
```

output:-



```
user@linux-OptiPlex-5090:~/Desktop/1SI19CS144/cns$ g++ dssserver.cpp
user@linux-OptiPlex-5090:~/Desktop/1SI19CS144/cns$ ./a.out

Enter port : 4444

Server Online. Waiting for client....
Connection Established.

Enter a large prime number, p : 13
Enter a prime number, q (p-1 divisible by q & q>2) : 3
Enter message, M = 243

H(M) = 1057
g    = 9
Server's Private key, x = 2
Server's Public  key, y = 3
Secret key, k = 2

Server's Signature {r,s} = {0, 2}

Sent p, q, g, and public key to client.
Sent message along with signature to client.

user@linux-OptiPlex-5090:~/Desktop/1SI19CS144/cns$
```



```
user@linux-OptiPlex-5090:~/Desktop/1SI19CS144/cns$ g++ dssclient.cpp
user@linux-OptiPlex-5090:~/Desktop/1SI19CS144/cns$ ./a.out

Enter server's IP address: 127.0.0.1
Enter port : 4444

Client is connected to Server.
Received p =  13
Received q =  3
Received g =  9
Received y =  3
Received M'=  243
Received r' = 0
Received s' = 2

H(M') = 1057
w = 2
v = 0

Digital Signature Verified.

user@linux-OptiPlex-5090:~/Desktop/1SI19CS144/cns$
```