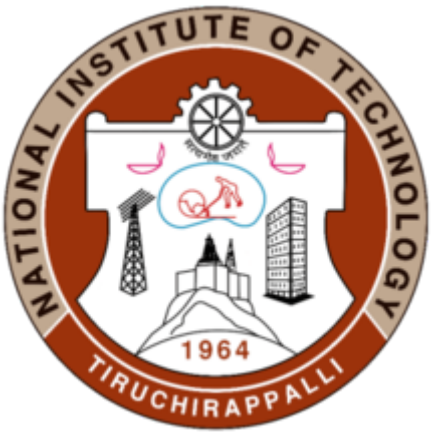
National Institute of Technology, Tiruchirappalli



Department of Computer Applications

Information Security Lab

Lab 4

Submitted to:

Dr. Ghanshyam Bopche

Submitted by:

Akriti Upadhyay

205120007

MCA – 2nd Year

Asymmetric Key Cryptosystem Algos

1. RSA:

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

int n, z, e, d;
// Vector to store ASCII values of characters from A-Z
vector<int> P;



// GCD(a,b)
int gcd(int a, int b)
{
    return b == 0 ? a : gcd(b, a % b);
}

// Print the registers
void printRegister(vector<int> reg){
    for(int i=0; i<reg.size(); i++)
        cout<<reg[i]<<" ";
    cout<<"\n";
}


// ENCRYPTION
string encryption(string plain, int p, int q)
{
    // STEP 2:
    n = p*q;

    // STEP 3:
    z = (p-1)*(q-1);

    // STEP 4:
    // choose (e) relatively prime to z (ie, GCD(e, z) = 1)
    for(int i=2;i<z;i++){
        if(gcd(z,i) == 1)
        {
            e = i;
```

```
            break;
        }
    }

    // STEP 5:
    // find d (multiplicative inverse of e % z), ie, (e*d)%z = 1
    for(int i=2;;i++){
        if((e*i)%z == 1)
        {
            d = i;
            break;
        }
    }

    // output values
    cout<<"\ne = "<<e<<",    "<<"n = "<<n<<",    "<<"d = "<<d<<endl;


    // Encrypting
    // Public key: {n, e}
    // C = P^e % n
    string result, res, cipher;

    // converting text form of plain text chars to ASCII equivalent
(converting range 65-90 to 0-25)
    for(int i=0; i<plain.length(); i++)
    {
        // if char is b/w A-Z
        if((int)plain[i]>64 && (int)plain[i]<91)
            // bringing value in range 0-25
            res.push_back(plain[i]-65);
        // keep space as it is
        if(plain[i]==' ')
            res.push_back(' ');
    }


    // encryption process => P^e % n
    for(int i=0; i<res.length(); i++)
    {
        long long z = 1; // P^e
        if(res[i] == ' ')
        {
```

```
            result.push_back(' ');
            P.push_back((int)res[i]);
        }
        else
        {
            // P^e
            for(int j=0; j<e; j++)
                z = ((int)res[i]) * z;
            // store in P to differentiate b/w numbers like 7, 33,
where 7%26 = 33%26 = 7 (storing 1 in P, if z%n > 26 | storing 0, if z%n
< 26)
            P.push_back((z % n)/26);
            result.push_back((z % n)%26);
        }
    }


    // changing ASCII values to alphabetic characters (converting range
0-25 to 65-90)
    for(int i=0; i<result.length() ;i++)
    {
        if((int)result[i]>=0 && (int)result[i]<26)
            cipher.push_back(result[i]+65);
        if(result[i]==' ')
            cipher.push_back(' ');
    }
    return cipher;
}



// DECRYPTION
string decryption(string cipher)
{
    // Decryption
    // Private key: {n, d}
    // P = C^d % n

    string result, res, final;

    // converting text form of cipher text chars to ASCII equivalent
(converting range 65-90 to 0-25)
    for(int i=0; i<cipher.length(); i++)
```

```
    {
        if(cipher[i]>64 && cipher[i]<91)
            // add P[i]*26 if > 26
            res.push_back((cipher[i]-65) + (P[i]*26));
        if(cipher[i] == ' ')
            res.push_back(' ');
    }

    // decryption process => C^d % n
    for(int i=0; i<res.length(); i++)
    {
        long long z = 1; // C^d
        if(res[i] == ' ')
            result.push_back(' ');
        else
        {
            // C^d
            for(int j=0; j<d; j++)
                z = ((int)res[i]) * z;
            result.push_back(z % n);
        }
    }

    // changing ASCII values to alphabetic characters (converting range
0-25 to 65-90)
    for(int i=0; i<result.length(); i++)
    {
        if((int)result[i]>=0 && (int)result[i]<26)
            final.push_back(result[i] + 65);
        if(result[i] == ' ')
            final.push_back(' ');
    }

    return final;
}



int main()
{
    // Taking plain text from user
    string plain;
    cout<<"\nEnter Plain Text: "<<endl;
```

```
    getline(cin,plain);


    // STEP 1:
    // take large prime numbers: p, q
    int p;
    int q;
    cout<<"\nEnter two prime numbers p and q (such that, p*q < 40):
\n";
    cin>>p>>q;

    // Encryption
    string cipher = encryption(plain,p,q);
    cout<<"\nCipher Text: "<<cipher<<endl;

    // Decryption
    string decrypt = decryption(cipher);
    cout<<"Decrypted Plain Text: "<<decrypt<<"\n\n";

    return 0;
}
```

```
Enter Plain Text:
AKRITI UPADHYAY

Enter two prime numbers p and q (such that, p*q < 40):
3
11

e = 3,   n = 33,   d = 7

Cipher Text: AKDRCR OJABNEAE
Decrypted Plain Text: AKRITI UPADHYAY
```

---

2. Diffie-Hellman:

```
#include<iostream>
using namespace std;

// Array of first 26 prime numbers
// A-Z => 26 alphabets (only Uppercase)
```

```
// Pick value of 'g' according to the ASCII value of alphabet character
int
g[26]={2,3,5,7,11,13,17,19,23,29,31,37,41,43,47,53,59,61,67,71,73,79,83
,89,97,101};


void diffie_helman(string key, int pvtA, int pvtB, int prime)
{
    string a, b, sendToB, sendToA;
    // sendToB = g^x % n
    // b = sendToB ^ y => (g^x % n)^y => shared key of B
    // sendToA = g^y % n
    // a = sendToA ^ x => (g^y % n)^x => shared key of A


    // computing, sendToB = g^x % n
    for(int i=0; i<key.length(); i++)
    {
        long long z = 1;  // z = g^x
        // keep spaces as it is
        if(key[i] == ' ')
            sendToB.push_back(' ');
        else
        {
            // if ascii(char) is in range A-Z
            if(key[i]>64 && key[i]<91){
                // g^x
                for(int j=0; j<pvtA; j++)
                    z = z * g[key[i]-65];
            }
            sendToB.push_back(z % prime);
        }
    }

    // computing b = sendToB ^ y
    for(int i=0; i<sendToB.length(); i++)
    {
        long long z = 1;
        // keep space as it is
        if(sendToB[i] == ' ')
            b.push_back(' ');
        else
        {
```

```
            // sendTOB ^ y
            for(int j=0; j<pvtB; j++)
                z = z * sendToB[i];
            // ascii to alphabet
            b.push_back(((z % prime) % 26) + 65);
        }
    }

    // computing, sendToA = g^y % n
    for(int i=0; i<key.length(); i++)
    {
        long long z = 1;
        if(key[i] == ' ')
            sendToA.push_back(' ');
        else
        {
            if(key[i]>64 && key[i]<91){
                // g^y
                for(int j=0; j<pvtB; j++)
                    z = z * g[key[i]-65];
            }
            sendToA.push_back(z % prime);
        }
    }

    // computing a = sendToA ^ x
    for(int i=0; i<sendToA.length(); i++)
    {
        long long z = 1;
        if(sendToA[i] == ' ')
            a.push_back(' ');
        else
        {

            for(int j=0; j<pvtA; j++)
                z = z*sendToA[i];
            // ascii to alphabet
            a.push_back(((z % prime) % 26) + 65);
        }
    }
    cout<<"\nFinal shared key of A: "<<a<<endl;
    cout<<"Final shared key of B: "<<b<<endl;
    if(a == b)
```

```
        cout<<"Shared keys of A and B are equal\n\n";
    else
        cout<<"Shared keys of A and B are NOT equal\n\n";
}

int main()
{
    // Input the encrypted key
    string key;
    // pvtA - private key of A (x)
    // pvtB - private key of B (y)
    int pvtA, pvtB;
    cout<<"\nEnter the key to be exchanged (UPPERCASE ALPHABETS ONLY):
\n";
    getline(cin,key);

    // n - prime num
    int n;
    // Bcoz max. value we can handle is 10^100 (otherwise overflow in
datatypes)
    cout<<"\nEnter a prime number (less than 100)"<<endl;
    cin>>n;


    cout<<"\nEnter the private keys of A and B, \nNOTE: \n1.
priv_key(A) < priv_key(B)\n2. Both the private keys should be <=
10)\n";
    cout<<"\nEnter private key of A: \n";
    cin>>pvtA;    // x
    cout<<"\nEnter private key of B: \n";
    cin>>pvtB;    // y

    // Key distribution algorithm
    diffie_helman(key, pvtA, pvtB, n);

    return 0;
}
```

```
Enter the key to be exchanged (UPPERCASE ALPHABETS ONLY):
SECRET KEY

Enter a prime number (less than 100)
19

Enter the private keys of A and B,
NOTE:
1. priv_key(A) < priv_key(B)
2. Both the private keys should be <= 10)

Enter private key of A:
6

Enter private key of B:
4

Final shared key of A: LBHLBH BBH
Final shared key of B: LBHLBH BBH
Shared keys of A and B are equal
```

X-X-X-X