Assignment-10

Q1. Write a program to implement the Knapsack algorithm.

A:

Pseudocode and Explanation –

1. **Modular Inverse Calculation (modInverse)**: Computes the modular inverse of w modulo M using the Extended Euclidean Algorithm. This inverse is used in decryption to reverse the effect of the multiplier w.

2. **Input Message and Keys**:

- Takes a binary message (as a string) and splits it into groups.
- Accepts a **private key** (superincreasing sequence) and calculates the **public key** by multiplying each private key value by w modulo M.

3. **Encryption**:

- Converts each group of bits to a sum of **public key values** based on the presence of '1' in each position.
- Stores these sums in cipher_text as encrypted values.

4. **Decryption**:

- Multiplies each encrypted sum by w_inverse modulo M to undo the effect of w.
- Recovers the original binary message by checking against the private key values in reverse, recreating each bit.

Code –

```
#include<bits/stdc++.h>
using namespace std;
int modInverse(int w, int M) {
    int m0 = M, y = 0, x = 1;
    if (M == 1)
        return 0;

    while (w > 1) {
```

```
        int q = w / M;
        int t = M;


        M = w % M;
        w = t;
        t = y;


        y = x - q * y;
        x = t;
    }


    if (x < 0)
        x += m0;

    return x;
}
int main()
{
    string pt;
    cout<<"Enter the message: "<<endl;
    cin>>pt;

    int grps;
    cout<<"Enter the number of groups in knapsack: "<<endl;
    cin>>grps;

    vector<int> private_key;
    cout<<"Enter the private key: "<<endl;
    for(int i=0;i<grps;i++){
        int temp;
        cin>>temp;
        private_key.push_back(temp);
    }

    int M,w;
    cout<<"Enter M and w such that they are coprime: "<<endl;
    cin>>M>>w;

    vector<int> public_key;
    for(int i=0;i<grps;i++){
        int val = private_key[i]*w;
        val = val%M;
        public_key.push_back(val);
    }
```

```cpp
    // Encyption
    vector<int> cipher_text;
    for(int i=0;i<pt.length();i+=grps){
        string temp = pt.substr(i,grps);
        int sum = 0;
        for(int j=0;j<temp.length();j++){
            if(temp[j] == '1'){
                sum+=public_key[j];
            }
        }
        cipher_text.push_back(sum);
    }

    for(int i=0;i<cipher_text.size();i++){
        cout<<cipher_text[i]<<" ";
    }

    cout<<endl;
    // Decryption

    int w_inverse = modInverse(w,M);

    string decrypted_message = "";
    for (int c : cipher_text) {

        int sum = (c * w_inverse) % M;
        string group_bits = "";

        for (int i = grps - 1; i >= 0; i--) {
            if (sum >= private_key[i]) {
                sum -= private_key[i];
                group_bits = '1' + group_bits;
            } else {
                group_bits = '0' + group_bits;
            }
        }
        decrypted_message += group_bits;
    }

    cout<<"Decrypted Message: "<<decrypted_message<<endl;

}
```

Output –

```
($?) { .\knapsack }
Enter the message:
100100111100101110
Enter the number of groups in knapsack:
6
Enter the private key:
1
2
4
10
20
40
Enter M and w such that they are coprime:
110 31
121 197 205
Decrypted Message: 100100111100101110
```

Q2. Write a program to implement the Elgamal algorithm.

A:

Pseudocode and Explanation –

This code demonstrates the **ElGamal cryptosystem** for encrypting and decrypting a message. It starts with fixed values for the prime p=11, generator =2, and private key x=3, computing the public key y=g^x mod p. For encryption, a message m is combined with an ephemeral key k=4 to produce ciphertext pair (c1,c2), where c1=g^k mod p and c2=m*y^k mod p. Decryption uses the private key x and modular inverses to recover mmm from (c1,c2). The program outputs public/private keys, encrypted, and decrypted messages, showing the basic mechanics of ElGamal encryption.

Code –

```
#include<bits/stdc++.h>
using namespace std;
// Function to perform modular exponentiation: (base^exp) % mod
long long modExp(long long base, long long exp, long long mod) {
    long long result = 1;
    while (exp > 0) {
        if (exp % 2 == 1)
            result = (result * base) % mod;
        base = (base * base) % mod;
        exp /= 2;
    }
    return result;
}
```

```cpp
// ElGamal Key Generation
void keyGeneration(long long &p, long long &g, long long &y, long long &x) {

    p = 11;
    g = 2;
    x = 3;

    y = modExp(g, x, p);
}

// ElGamal Encryption
pair<long long, long long> encrypt(long long m, long long p, long long g, long
long y) {
    long long k = 4;
    long long c1 = modExp(g, k, p);
    long long c2 = (m * modExp(y, k, p)) % p;

    return make_pair(c1, c2);
}

// ElGamal Decryption
long long decrypt(pair<long long, long long> ciphertext, long long p, long
long x) {
    long long c1 = ciphertext.first;
    long long c2 = ciphertext.second;


    long long s = modExp(c1, x, p);

    long long s_inv = modExp(s, p - 2, p);

    long long m = (c2 * s_inv) % p;

    return m;
}

int main() {
    long long p, g, y, x;
    keyGeneration(p, g, y, x);

    cout << "Public Key (p, g, y): (" << p << ", " << g << ", " << y << ")" <<
endl;
    cout << "Private Key (x): " << x << endl;

    long long message;
    cout << "Enter the message to encrypt (as an integer < p): ";
    cin >> message;

    auto ciphertext = encrypt(message, p, g, y);
```

```cpp
    cout << "Encrypted Message (c1, c2): (" << ciphertext.first << ", " <<
ciphertext.second << ")" << endl;

    long long decryptedMessage = decrypt(ciphertext, p, x);
    cout << "Decrypted Message: " << decryptedMessage << endl;

    return 0;
}
```

Output –

```
PS C:\Users\arinr\Desktop\Crypto_Lab\Lab_10> cd "c:\Users\arinr\Desktop\Crypto_Lab\Lab_10\" ; if ($?) { g++ elgamal.cpp -o elgamal } ; if ($
?) { .\elgamal }
Public Key (p, g, y): (11, 2, 8)
Private Key (x): 3
Enter the message to encrypt (as an integer < p): 7
Encrypted Message (c1, c2): (5, 6)
Decrypted Message: 7
PS C:\Users\arinr\Desktop\Crypto_Lab\Lab_10>
```