Experiment No. 5

Title: Knapsack Cipher

Batch: B2 **Roll No.: 16010420117** **Experiment No.: 5**

Title: To implement Knapsack Cipher.

Resources needed: Windows/Linux OS

Theory:

The Merkle-Hellman knapsack cryptosystem was proposed by Merkle and Hellman. Merkle was also one of the founders of public key cryptography. The Merkle-Hellman knapsack cryptosystem is based on a problem that is known to be NP-complete. This seems to make it an ideal candidate for a secure public key cryptosystem. The knapsack problem can be stated as follows.
Given a set of n weights W0,W1, . . . , Wn−1 and sum S, find a0, a1, . . . , an−1, where each ai ϵ {0, 1}, so that S = a0W0 + a1W1 +…+an−1Wn−1 will form the general knapsack.
A super increasing knapsack, is similar to the general knapsack except that when the weights are arranged from least to greatest, each weight is greater than sum of all previous weights

---

Algorithm :

A. Key pair generation :

1. Generate a super increasing knapsack S=(S1,S2,…,Sr).
2. Convert the super increasing knapsack into a general knapsack G=(G1,G2,…,Gr).
3. Choose a multiplier *m* and a modulus *n* so that *m* and *n* are relatively prime and *n* is greater than the sum of all elements in the super increasing knapsack.
4. Then the general knapsack is computed from the super increasing knapsack by modular multiplication: Gi=Si*m mod n
5. The public key is the general knapsack.
6. The private key is the super increasing knapsack together with the conversion factors

B. Encryption:

1. First convert plain text message M into binary equivalent $(M1,M2,\ldots,Mn)_2$.
2. Calculate C=M1G1+M2G2+…….GnMn
3. Display C

C. Decryption:

1. Calculate m^{-1} (multiplicative inverse of m under modulus n)
2. Calculate P'=C m^{-1} mod n
3. Select weights from super increasing knapsack starting from largest weight so that it will sum up to P'.
4. This will give the binary equivalent of the plaintext, convert it to decimal equivalent and display.

Example :

1. We'll choose the superincreasing knapsack
   (2, 3, 7, 14, 30, 57, 120, 251).

2. To convert the superincreasing knapsack into a general knapsack, we choose a multiplier *m* and a modulus *n* so that *m* and *n* are relatively prime and *n* is greater than the sum of all elements in the superincreasing knapsack. For this example, we choose *m* = 41 and *n* = 491. Then the general knapsack is computed from the superincreasing knapsack by modular multiplication:

   $2m = 2 * 41 = 82 \bmod 491$

   $3m = 3 * 41 = 123 \bmod 491$

   $7m = 7 * 41 = 287 \bmod 491$

   $14m = 14 * 41 = 83 \bmod 491$

   $30m = 30 * 41 = 248 \bmod 491$

   $57m = 57 * 41 = 373 \bmod 491$

   $120m = 120 * 41 = 10 \bmod 491$

   $251m = 251 * 41 = 471 \bmod 491$

   The resulting general knapsack is *(*82, 123, 287, 83, 248, 373, 10, 471*)*, which appears to be a general (non-super increasing) knapsack.

3. The public key is the general knapsack
   Public key: *(*82, 123, 287, 83, 248, 373, 10, 471*).*

4. The private key is the super increasing knapsack together with the modular inverse of the conversion factor *m*, that is,
   Private key: *(*2, 3, 7, 14, 30, 57, 120, 251*)* and

   $m^{-1} \bmod n = 41^{-1} \bmod 491 = 12.$

Suppose Bob's public and private key pair are given in *step* 3 and *step* 4, above. Then if Alice wants to encrypt the message *M* = 150 for Bob,

1. To encrypt, Alice first converts 150 to binary, that is, 10010110. Then she uses the 1 bits to select the elements of the general knapsack that are summed to give the ciphertext. In this case, Alice finds

   $C = 82 + 83 + 373 + 10 = 548.$

2. To decrypt this ciphertext, Bob uses his private key to find

   $Cm^{-1} \bmod n = 548 * 12 \bmod 491 = 193$

3. Bob then solves the super increasing knapsack for 193. This is an easy (linear time) problem from which Bob recovers the message in binary 10010110 or, in decimal, $M = 150$.

Results: (Program printout with output / Document printout as per the format)

Code:

```
superSeq = [3, 5, 9, 20, 44]

publicKey = []

m = 67

n = 89


for i in superSeq:

    x = (i*n) % m

    publicKey.append(x)


print(f"\nPublic Key:\t{publicKey}")

print(f"\nPrivate Key:\t{superSeq}")


def knapsackEncrypt(plainText, superSeq, n, m):

    cipherText = []

    index = 0

    sum = 0


    for bit in plainText:

        if bit == '1':

            sum += publicKey[index]

        index += 1

        if index == 5:

            cipherText.append(sum)

            index = 0
```

```
            sum = 0
    return cipherText


def inverse(a, m):
    for x in range(1, m):
        if (((a % m) * (x % m)) % m == 1):
            return x
    return -1


def knapsackDecrypt(ciphertext, superSeq, n, m):
    inversed = inverse(n, m)
    decodedValues = [(x * inversed) % m for x in cipherText]
    sum = 0
    plainText = ''

    for value in decodedValues:
        ans = []
        for item in superSeq[::-1]:
            if sum + item <= value:
                sum += item
                ans.append('1')
            else:
                ans.append('0')

        plainText += ''.join(ans[::-1])
        sum = 0

    return plainText
```

```
plainText = input('Enter plaintext bits: ')

binChunks = [plainText[6*i:6*(i+1)] for i in range(len(plainText)//6)]

print(f"\nSplitted plain text:\t{binChunks}")


cipherText = knapsackEncrypt(plainText, superSeq, n, m)

decodedPlainText = knapsackDecrypt(cipherText, superSeq, n, m)

print(f'CipherText:\t{cipherText}')

print(f'\nPlainText:\t{decodedPlainText}')
```

Output:

```
Public Key:     [66, 43, 64, 38, 30]

Private Key:    [3, 5, 9, 20, 44]
Enter plaintext bits: 000011010011000011010111010110

Splitted plain text:    ['000011', '010011', '000011', '010111', '010110']
CipherText:     [30, 130, 109, 137, 145, 168]

PlainText:      000011010011000011010111010110
```

Questions:

1. Explain Digital signature using Knapsack with suitable example.

Ans: If Alice wants to sign a message M, she should perform the signature procedure as follows.

Step 1. Choose a random number k ∈ [1, n − 1]. Step 2. Compute the signature (r, s) of the message M with r = M + (kG)x mod n, and s = k − H(r)xAlice mod n. Step 3. Apply the knapsack value for the signed message: 1) R = Knapsack(r) and S = Knapsack(s). 2) Send a signed message {R, S, IDAlice} to user Bob.
Message Recovery Phase: After receiving {R, S, IDAlice}, Bob can apply the reverse knapsack. Recover and
verify the message M with M = r − (sG + H(r)(yAlice)x mod n.

2. Explain any one attack on Knapsack.

Ans: In 1984 Adi Shamir published an attack on the Merkle-Hellman cryptosystem which can decrypt encrypted messages in polynomial time without using the private key. The attack analyzes the public key B=(b1, b2, …bn) and searches for a pair of numbers u and c such that ubi mod m is a superincreasing sequence. The (u,m) pair found by the attack may not be equal to (r',q) in the private key, but like that pair it can be used to transform a hard knapsack problem using B into an easy problem using a superincreasing sequence. The attack operates solely on the public key; no access to encrypted messages is necessary.

Outcomes: CO 2: Illustrate different cryptographic algorithms for security.

Conclusion : Merkle–Hellman knapsack cipher (knapsack cipher) was studied, observed, understood and implemented successfully.

Grade: AA / AB / BB / BC / CC / CD /DD

Signature of faculty in-charge with date

References:

Books/ Journals/ Websites:

- Mark Stamp, “Information security Principles and Practice” Wiley