

**Experiment No. 5 Title:** Knapsack Cipher

**Batch: B1 Roll No.: 1914078 Experiment No.: 5**

**Aim:** To implement Knapsack Cipher

**Resources needed:** Windows/Linux, C or JAVA language

# Theory: Pre Lab/ Prior Concepts:

Knapsack Encryption Algorithm is the first general public key cryptography algorithm. It is developed by Ralph Merkle and Mertin Hellman in 1978. As it is a Public key cryptography, it needs two different keys.

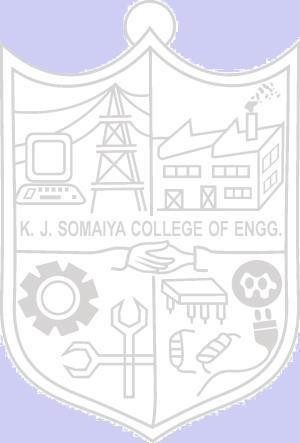
The general knapsack problem can be stated as: Given a set of n weights, W1,W2,...,Wn, and a sum S, is it possible to find ai∈{0,1} such that

S = a1W1+a2W2+...+ anWn

The general knapsack problem is NP-complete.

However, a superincreasing knapsack, i.e., a knapsack in which each weight is greater than the sum of all previous weights, is easy to solve.

A public key crpytosystem can be constructed based on the knapsack problem by

1. Generate a superincreasing knapsack.
2. Convert the superincreasing knapsack into a "general" knapsack by multiplying all of the numbers by n mod m, where m is greater than the sum of all of the numbers in the knapsack and the n has no factor in common with m.
3. The superincreasing knapsack, together with m and n, is the private key.
4. The corresponding general knapsack is the public key.
5. It is easy to encrypt using the general knapsack; simply add all of the elements in the 1's positions.
6. Given the private key, the ciphertext can be converted to a superincreasing knapsack, which is then easily solved to decrypt the message.
7. Without the private key, an attacker must solve the "general" knapsack problem to recover the message.

# Procedure / Approach /Algorithm / Activity Diagram:

To produce a normal knapsack sequence, take a superincreasing sequence; e.g. {1,2, 4, 10, 20, 40}. Multiply all the values by a number, n, modulo m. The modulus should be a number greater than the sum of all the numbers in the sequence, for example, 110. The multiplier should have no factors in common with the modulus. So let's choose 31. The normal knapsack sequence would be:

1×31 mod(110) = 31

2×31 mod(110) = 62

4×31 mod(110) = 14

10×31 mod(110) = 90

20×31 mod(110) = 70

40×31 mod(110) = 30

So the public key is: {31, 62, 14, 90, 70, 30} and

the private key is {1, 2, 4, 10, 20.40}.

Let's try to send a message that is in binary code:

100100111100101110

The knapsack contains six weights so we need to split the message into groups of six: 100100

111100

101110

This corresponds to three sets of weights with totals as follows 100100 = 31 + 90 = 121

111100 = 31+62+14+90 = 197

101110 = 31+14+90+70 =205

So the coded message is 121 197 205.

Now the receiver has to decode the message... The person decoding must know the two numbers 110 and 31 (the modulus and the multiplier). Let's call the modulus "m" and the number you multiply by "n". We need *n*−1, which is a multiplicative inverse of n mod m, i.e. *n n*−1 = 1 mod m

In this case, *n*−1to be 71.

All you then have to do is multiply each of the codes 71 mod 110 to find the total in the knapsack which contains {1, 2, 4, 10, 20, 40} and hence to decode the message.

The coded message is 121 197 205:

121×71 mod(110) = 11 = 100100

197×71 mod(110) = 17 = 111100

205×71 mod(110) = 35 = 101110

The decoded message is:

100100111100101110.

# Activity:

1. Perform Knapsack Crypto algorithm using { 3,5,9,20,44 } where m=89 and n=67. .
2. Perform encryption and decryption the message BUY NOW

{000011010011000011010111010110}

1. What is the public key, private key, plain text and cipher text?

**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}")

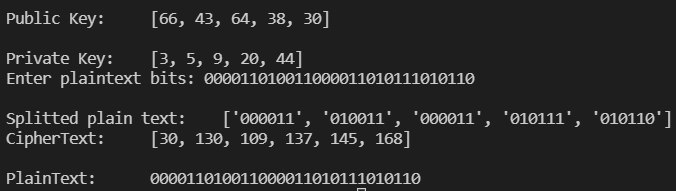
cipherText = knapsackEncrypt(plainText, superSeq, n, m)

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

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

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

**Output:**



# Questions:

1. What is the drawback of knapsack cipher?

**Answer:**

**The drawbacks of knapsack cipher are-**

1. The length of the keys become a burden to the users of the system.
2. 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 analyses the public key B = {b1, b2, …, bn} and searches for a pair of numbers u and m such that (ubi mod m) is a superincreasing sequence.
3. The (u,m) pair found by the attack may not be equal to 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.
4. What are the applications of knapsack cipher?

**Answer:**

Knapsack cipher is a public key crypto system which utilizes the knapsack

problem to encrypt messages.

1. **Trapdoor**: Convert Super increasing knapsack into “general” knapsack using modular arithmetic
2. **One-way**: General knapsack is easy to encrypt but hard to solve whereas the Super increasing knapsack is easy to solve and is kept hidden making it secure.

# Use of public-key cryptosystems can be classified into three categories

* **Encryption /decryption**: The sender encrypts a message with the recipient’s public key.
* **Digital signature**: The sender “signs” a message with its private key. Signing is achieved by a cryptographic algorithm applied to the message or to a small block of data that is a function of the message.
* **Key exchange**: Two sides cooperate to exchange a session key. Several different approaches are possible, involving the private key(s) of one or both parties.

**Public Key Cryptography is used in a number of applications and systems software.**

* Digitally signed document
* E-mail encryption software such as PGP and MIME
* SSL protocol
* SSH protocol

**Outcome:**

**CO2:** Illustrate different cryptographic algorithms for security.

**Conclusion:** We learnt how to implement knapsack cryptography algorithm and wrote a Python code for the same.

# Grade: AA / AB / BB / BC / CC / CD /DD Signature of faculty in-charge with date

**References: Books/ Journals/ Websites:**

1. Charles P. Pfleeger, “Security in Computing”, Pearson Education
2. Behrouz A. Forouzan, “Cryptography and Network Security”, Tata McGraw Hill
3. William Stalling, “Cryptography and Network Security”, Prentice Hall
4. <http://www.cs.sjsu.edu/~stamp/CS265/SecurityEngineering/chapter5_SE/knapsack.html>