16. def printString(S, N):

plaintext = [None] \* 5

freq = [0] \* 26

freqSorted = [None] \* 26

used = [0] \* 26

for i in range(N):

if S[i] != ' ':

freq[ord(S[i]) - 65] += 1

for i in range(26):

freqSorted[i] = freq[i]

T = "ETAOINSHRDLCUMWFGYPBVKJXQZ"

freqSorted.sort(reverse = True)

for i in range(5):

ch = -1

for j in range(26):

if freqSorted[i] == freq[j] and used[j] == 0:

used[j] = 1

ch = j

break

if ch == -1:

break

x = ord(T[i]) - 65

x = x - ch

curr = ""

for k in range(N):

if S[k] == ' ':

curr += " "

continue

y = ord(S[k]) - 65

y += x

if y < 0:

y += 26

if y > 25:

y -= 26

curr += chr(y + 65)

plaintext[i] = curr

for i in range(5):

print(plaintext[i])

S = "B TJNQMF NFTTBHF"

N = len(S)

printString(S, N)

output:

A SIMPLE MESSAGE

B TJNQMF NFTTBHF

A SIMPLE MESSAGE

C UKORNG OGUUCIG

C UKORNG OGUUCIG

17. import java.math.BigInteger;

import java.security.SecureRandom;

public class RSA {

private final static BigInteger one = new BigInteger("1");

private final static SecureRandom random = new SecureRandom();

private BigInteger privateKey;

private BigInteger publicKey;

private BigInteger modulus;

public RSA(int bitLength) {

BigInteger p = BigInteger.probablePrime(bitLength / 2, random);

BigInteger q = BigInteger.probablePrime(bitLength / 2, random);

modulus = p.multiply(q);

BigInteger phi = (p.subtract(one)).multiply(q.subtract(one));

publicKey = new BigInteger("65537"); // Common public exponent

privateKey = publicKey.modInverse(phi);

}

public BigInteger encrypt(BigInteger message) {

return message.modPow(publicKey, modulus);

}

public BigInteger decrypt(BigInteger encryptedMessage) {

return encryptedMessage.modPow(privateKey, modulus);

}

public static void main(String[] args) {

RSA rsa = new RSA(1024);

BigInteger message = new BigInteger("1234567890");

// Encryption

BigInteger encryptedMessage = rsa.encrypt(message);

System.out.println("Encrypted message: " + encryptedMessage);

// Decryption

BigInteger decryptedMessage = rsa.decrypt(encryptedMessage);

System.out.println("Decrypted message: " + decryptedMessage);

}

}

18. import random

# function to check if a number is prime

def is\_prime(num):

if num == 2:

return True

if num < 2 or num % 2 == 0:

return False

for n in range(3, int(num \*\* 0.5) + 2, 2):

if num % n == 0:

return False

return True

# function to compute the gcd of two numbers

def gcd(a, b):

while b != 0:

a, b = b, a % b

return a

# function to generate the public and private keys

def generate\_keys():

# generate two large prime numbers

p = random.randint(1000, 10000)

while not is\_prime(p):

p += 1

q = random.randint(1000, 10000)

while not is\_prime(q) or q == p:

q += 1

# compute n and phi(n)

n = p \* q

phi\_n = (p-1) \* (q-1)

# choose a large random exponent e that is relatively prime to phi(n)

e = random.randint(10000, 100000)

while gcd(e, phi\_n) != 1:

e += 1

# compute the modular multiplicative inverse of e modulo phi(n)

d = pow(e, -1, phi\_n)

return (e, n), (d, n)

# function to encrypt a message

def encrypt(public\_key, message):

e, n = public\_key

cipher = []

for m in message:

c = pow(ord(m)-65, e, n)

cipher.append(c)

return cipher

# function to decrypt a message

def decrypt(private\_key, cipher):

d, n = private\_key

message = ''

for c in cipher:

m = pow(c, d, n)

message += chr(m+65)

return message

# example usage

public\_key, private\_key = generate\_keys()

print("Public key:", public\_key)

print("Private key:", private\_key)

message = "HELLO WORLD"

cipher = encrypt(public\_key, message)

print("Encrypted message:", cipher)

decrypted\_message = decrypt(private\_key, cipher)

print("Decrypted message:", decrypted\_message)

output:

public key (34889,14802069)

private key(7950309,14802061)

encrypted message(4658137,13310228,717013)

19. #include <math.h>

#include <stdio.h>

long long int power(long long int a, long long int b,

long long int P)

{

if (b == 1)

return a;

else

return (((long long int)pow(a, b)) % P);

}

int main()

{

long long int P, G, x, a, y, b, ka, kb;

P = 23;

printf("The value of P : %lld\n", P);

G = 9;

printf("The value of G : %lld\n\n", G);

a = 4;

printf("The private key a for Alice : %lld\n", a);

x = power(G, a, P);

b = 3;

printf("The private key b for Bob : %lld\n\n", b);

y = power(G, b, P);

ka = power(y, a, P);

kb = power(x, b, P);

printf("Secret key for the Alice is : %lld\n", ka);

printf("Secret Key for the Bob is : %lld\n", kb);

return 0;

}

Output:

The value of P : 23

The value of G : 9

The private key a for Alice : 4

The private key b for Bob : 3

Secret key for the Alice is : 9

Secret Key for the Bob is : 9

20. cipher\_text = "53‡‡†305))6\*;4826)4‡.)4‡);806\*;48†8¶60))85;;]8\*;:‡8†83 (88)5†;46(;88\*96\*?;8)‡(;485);5†2:‡(;4956\*2(5—4)8¶8\*;4069285);)6†8)4‡‡;1(‡9;48081;8:8‡1;48†85;4)485†528806\*81 (‡9;48;(88;4(‡?34;48)4‡;161;:188;‡?;"

plain\_text = ""

mapping = {

'‡': 'a',

'†': 'e',

'¶': 'i',

'\*': 'o',

'(': 'u',

')': 'y',

';': ' ',

'—': '-',

']': ',',

':': '.',

'4': 't',

'5': 'h',

'8': 's',

'3': 'r',

'6': 'n',

'0': 'g',

'2': 'm',

'9': 'd',

'1': 'l',

'(': 'u',

'?': 'p',

'[': 'b',

'(': 'u',

'}': 'v',

'7': 'c',

}

for c in cipher\_text:

if c in mapping:

plain\_text += mapping[c]

else:

plain\_text += c

print(plain\_text)

output:

hraaerghyyno tsmnyta.ytay sgno tsesingyysh ,so .asesr ussyhe tnu ssodnop syau tshy hem.au tdhnomuh-tysiso tgndmshy ynesytaa luad tsgsl s.sal tsesh tytshehmssgnosl uad ts uss tuaprt tsyta lnl .lss ap

>