Dhaval Patel
M.Sc. (Cyber Security) Sem-3
Er no: 032200300002034

Q1. RSA encryption

RSA encryption Source code:-

```
#include<iostream>
#include<math.h>
#include<string.h>
#include<stdlib.h>

using namespace std;

long int p, q, n, t, flag, e[100], d[100], temp[100], j, m[100], en[100], i;
char msg[100];
int prime(long int);
void ce();
long int cd(long int);
void encrypt();
void decrypt();
int prime(long int pr)
{
int i;
j = sqrt(pr);
for (i = 2; i <= j; i++)
{
if (pr % i == 0)
return 0;
}
return 1;
}
int main()
{
cout << "\nEnter First Prime number : ";
cin >> p;
flag = prime(p);
if (flag == 0)
{
cout << "\nWRONG INPUT\n";
exit(1);
}
cout << "\nEnter another prime number : ";
cin >> q;
flag = prime(q);
if (flag == 0 || p == q)
```

```
{
cout << "\nWRONG INPUT\n";
exit(1);
}
cout << "\nEnter Message :";
fflush(stdin);
cin.getline(msg, 100);
for (i = 0; msg[i] != '\0'; i++)
m[i] = msg[i];
n = p * q;
t = (p - 1) * (q - 1);
ce();
cout << "\nPublic key (e, n): (" << *e << ", " << n << ")" << endl;
cout << "Private key (d): " << *d << endl;
encrypt();
decrypt();
return 0;
}
void ce()
{
int k;
k = 0;
for (i = 2; i < t; i++)
{
if (t % i == 0)
continue;
flag = prime(i);
if (flag == 1 && i != p && i != q)
{
e[k] = i;
flag = cd(e[k]);
if (flag > 0)
{
d[k] = flag;
k++;
}
if (k == 99)
break;
}
}
}
long int cd(long int x)
{
long int k = 1;
while (1)
{
k = k + t;
if (k % x == 0)
return (k / x);
}
```

```
}
void encrypt()
{
long int pt, ct, key = e[0], k, len;
i = 0;
len = strlen(msg);
while (i != len)
{
pt = m[i];
pt = pt - 96;
k = 1;
for (j = 0; j < key; j++)
{
k = k * pt;
k = k % n;
}
temp[i] = k;
ct = k + 96;
en[i] = ct;
i++;
}
en[i] = -1;
cout << "\nThe Encrypted Message is below : \n";
for (i = 0; en[i] != -1; i++)
printf("%c", en[i]);
}
void decrypt()
{
long int pt, ct, key = d[0], k;
i = 0;
while (en[i] != -1)
{
ct = temp[i];
k = 1;
for (j = 0; j < key; j++)
{
k = k * ct;
k = k % n;
}
pt = k + 96;
m[i] = pt;
i++;
}
m[i] = -1;
cout << "\n \nThe Decrypted Message is below : \n";
for (i = 0; m[i] != -1; i++)
printf("%c", m[i]);
cout<<endl;
}
```

Output:

```
"C:\Users\babbu\Documents\

Enter First Prime number : 37

Enter another prime number : 83

Enter Message :Hello Name : Babbu Rai, Er no : 032200300002031, M.Sc(Cyber Security) Sem-3

Public key (e, n): (5, 3071)
Private key (d): 1181

The Encrypted Message is below :
@û  ªÖ┘a5ûÖ7ÖÿaÇÇ□Ö╤a□─Ö╡τÖ∩ªÖ7ÖE¾ààEE¾EEEEàE¾╣─Ö≈┬ïS█¥─ÇûτÖïûS□τ□r─ƒÖïû5á¾Ö

The Decrypted Message is below :
Hello Name : Babbu Rai, Er no : 032200300002031, M.Sc(Cyber Security) Sem-3

Process returned 0 (0x0)   execution time : 129.143 s
Press any key to continue.
```

Q2. RSA digital signature implementation

Source code:

```
import random
from hashlib import sha256


def coprime(a, b):
while b != 0:
a, b = b, a % b
return a


def extended_gcd(aa, bb):
lastremainder, remainder = abs(aa), abs(bb)
x, lastx, y, lasty = 0, 1, 1, 0
while remainder:
lastremainder, (quotient, remainder) = remainder, divmod(lastremainder, remainder)
x, lastx = lastx - quotient*x, x
y, lasty = lasty - quotient*y, y
return lastremainder, lastx * (-1 if aa < 0 else 1), lasty * (-1 if bb < 0 else 1)

#Euclid's extended algorithm for finding the multiplicative inverse of two numbers
def modinv(a, m):
        g, x, y = extended_gcd(a, m)
        if g != 1:
                raise Exception('Modular inverse does not exist')
        return x % m


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


def generate_keypair(p, q):
if not (is_prime(p) and is_prime(q)):
raise ValueError('Both numbers must be prime.')
elif p == q:
raise ValueError('p and q cannot be equal')

n = p * q
```

```
#Phi is the totient of n
phi = (p-1) * (q-1)

#Choose an integer e such that e and phi(n) are coprime
e = random.randrange(1, phi)

#Use Euclid's Algorithm to verify that e and phi(n) are comprime
g = coprime(e, phi)

while g != 1:
e = random.randrange(1, phi)
g = coprime(e, phi)

#Use Extended Euclid's Algorithm to generate the private key
d = modinv(e, phi)

#Return public and private keypair
#Public key is (e, n) and private key is (d, n)
return ((e, n), (d, n))


def encrypt(privatek, plaintext):
#Unpack the key into it's components
key, n = privatek

#Convert each letter in the plaintext to numbers based on the character using a^b mod m

numberRepr = [ord(char) for char in plaintext]
print("Number representation before encryption: ", numberRepr)
cipher = [pow(ord(char),key,n) for char in plaintext]

#Return the array of bytes
return cipher


def decrypt(publick, ciphertext):
#Unpack the key into its components
key, n = publick

#Generate the plaintext based on the ciphertext and key using a^b mod m
numberRepr = [pow(char, key, n) for char in ciphertext]
plain = [chr(pow(char, key, n)) for char in ciphertext]

print("Decrypted number representation is: ", numberRepr)

#Return the array of bytes as a string
return ''.join(plain)


def hashFunction(message):
```

```
hashed = sha256(message.encode("UTF-8")).hexdigest()
return hashed


def verify(receivedHashed, message):
ourHashed = hashFunction(message)
if receivedHashed == ourHashed:
print("Verification successful: ", )
print(receivedHashed, " = ", ourHashed)
else:

print("Verification failed")
print(receivedHashed, " != ", ourHashed)


def main():
p = int(input("Enter a prime number (17, 19, 23, etc): "))
q = int(input("Enter another prime number (Not one you entered above): "))
#p = 17
#q=23


print("Generating your public/private keypairs now . . .")
public, private = generate_keypair(p, q)

print("Your public key is ", public ," and your private key is ", private)
message = input("Enter a message to encrypt with your private key: ")
print("")

hashed = hashFunction(message)

print("Encrypting message with private key ", private ," . . .")
encrypted_msg = encrypt(private, hashed)
print("Your encrypted hashed message is: ")
print(''.join(map(lambda x: str(x), encrypted_msg)))
#print(encrypted_msg)

print("")
print("Decrypting message with public key ", public ," . . .")

decrypted_msg = decrypt(public, encrypted_msg)
print("Your decrypted message is:")
print(decrypted_msg)

print("")
print("Verification process . . .")
verify(decrypted_msg, message)

main()
```

Output:

```
Windows PowerShell

PS C:\Users\babbu\OneDrive\Desktop> python rsa.py
Enter a prime number (17, 19, 23, etc): 19
Enter another prime number (Not one you entered above): 23
Generating your public/private keypairs now . . .
Your public key is  (137, 437)  and your private key is  (185, 437)
Enter a message to encrypt with your private key: hello

Encrypting message with private key  (185, 437)  . . .
Number representation before encryption:  [50, 99, 102, 50, 52, 100, 98, 97, 53, 102, 98, 48, 97, 51, 48, 101, 50, 54, 1
01, 56, 51, 98, 50, 97, 99, 53, 98, 57, 101, 50, 57, 101, 49, 98, 49, 54, 49, 101, 53, 99, 49, 102, 97, 55, 52, 50, 53,
101, 55, 51, 48, 52, 51, 51, 54, 50, 57, 51, 56, 98, 57, 56, 50, 52]
Your encrypted hashed message is:
312245296312200933824126829633898241356983473123083472273563383122412452683381934731219347643386430864347268245642962412
520031226834725356982003563563083121935622733819227312200

Decrypting message with public key  (137, 437)  . . .
Decrypted number representation is:  [50, 99, 102, 50, 52, 100, 98, 97, 53, 102, 98, 48, 97, 51, 48, 101, 50, 54, 101, 5
6, 51, 98, 50, 97, 99, 53, 98, 57, 101, 50, 57, 101, 49, 98, 49, 54, 49, 101, 53, 99, 49, 102, 97, 55, 52, 50, 53, 101,
55, 51, 48, 52, 51, 51, 54, 50, 57, 51, 56, 98, 57, 56, 50, 52]
Your decrypted message is:
2cf24dba5fb0a30e26e83b2ac5b9e29e1b161e5c1fa7425e73043362938b9824

Verification process . . .
Verification successful:
2cf24dba5fb0a30e26e83b2ac5b9e29e1b161e5c1fa7425e73043362938b9824  =  2cf24dba5fb0a30e26e83b2ac5b9e29e1b161e5c1fa7425e730
43362938b9824
PS C:\Users\babbu\OneDrive\Desktop>
```