Dhaval Patel

M.Sc. (Cyber Security) Sem-3

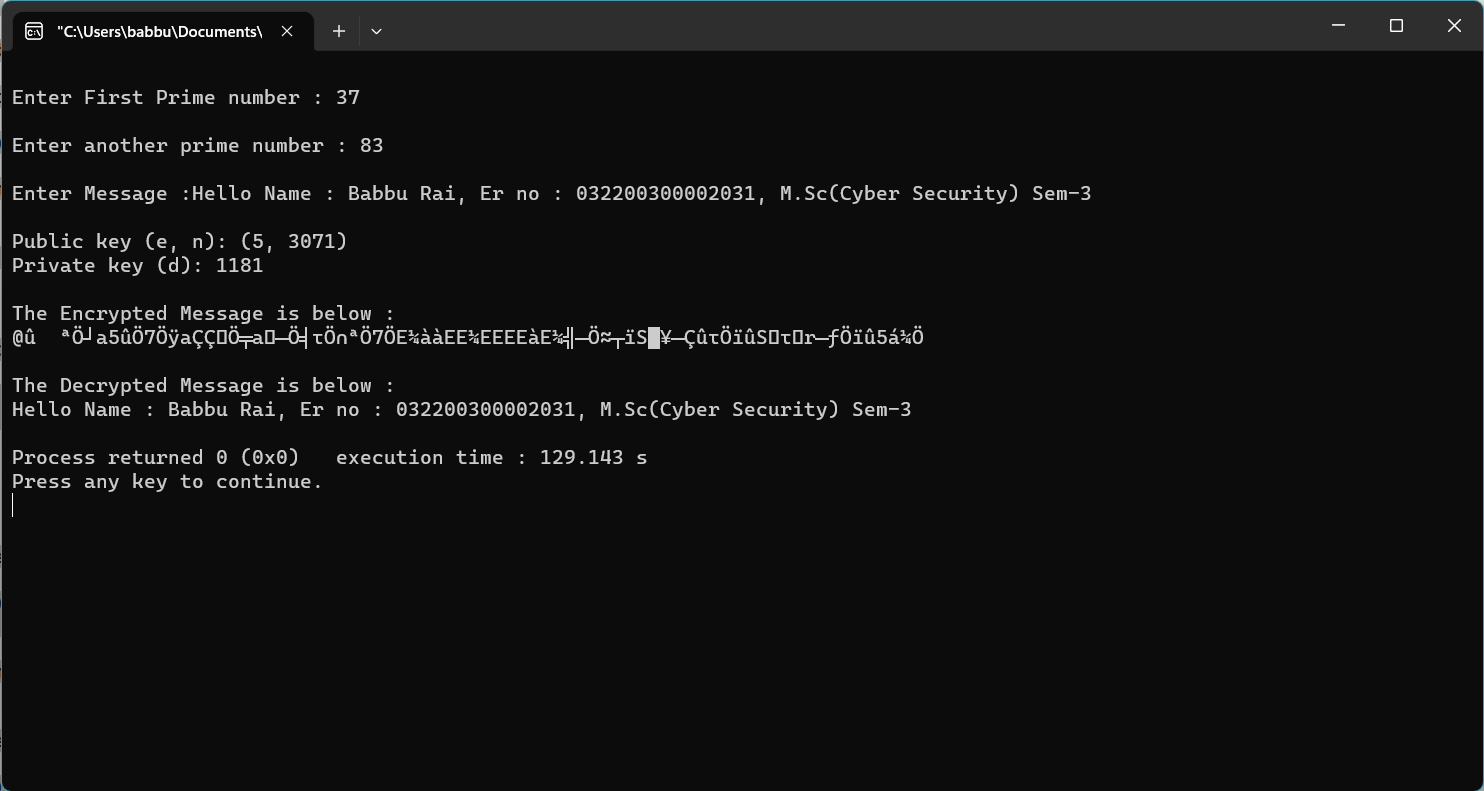
Er no: 032200300002034

# 

# 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: 

# 

# 

# 

# 

# 

# 

# 

# 

# 

# 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: 