Ex. No. 5	Asymmetric Cipher - RSA
Date of Exercise	31.08.2023

Aim

To Implement the Asymmetric Cipher – RSA

Description:

The RSA (Rivest–Shamir–Adleman) cryptosystem is a widely used and foundational public-key encryption algorithm in the field of modern cryptography. It was first introduced by its creators, Ron Rivest, Adi Shamir, and Leonard Adleman, in 1977. The RSA algorithm is named after their last names and is known for its robustness and security, particularly in securing digital communications and data.

RSA is an asymmetric encryption algorithm, meaning it uses a pair of keys: a public key for encryption and a private key for decryption. The security of RSA relies on the mathematical difficulty of factoring large composite numbers into their prime factors. The strength of RSA encryption is based on the impracticality of factoring the product of two large prime numbers into their constituent primes. This mathematical problem, known as the integer factorization problem, is challenging even for powerful computers and remains computationally infeasible for sufficiently large key sizes.

The RSA encryption process involves key generation, where a public key and a corresponding private key are created. The public key is used to encrypt data that can only be decrypted using the private key. This allows secure and confidential transmission of data over public networks, making RSA a vital component in securing online transactions, email communication, and digital signatures.

Despite its effectiveness, RSA encryption is computationally intensive, especially for large key sizes, making it slower compared to symmetric-key encryption algorithms for bulk data encryption. To address this, hybrid cryptosystems are often employed, where RSA is used to securely exchange a shared secret key for more efficient symmetric-key encryption.

RSA has had a profound impact on the field of cryptography and continues to be an essential component in ensuring data security in the digital age. Its versatility and robustness make it a standard choice for securing sensitive information and communications, from protecting online banking transactions to enabling secure access to digital identities and services.

Algorithm:

Step 1: Input: Prompt the user to enter two prime numbers, p and q, that serve as the building blocks for the RSA public and private keys.

Step 2: Calculate the RSA modulus n as the product of p and q.

Step 3: Calculate Euler's totient function value z as (p - 1) * (q - 1). This value is used in key generation.

Step 4: Define a function is_prime(num) to check if a number num is prime.

Step 5: Generate a random integer e between 2 and z - 1.

While e is not a prime number (determined using the is_prime function), regenerate e.

Step 6: Calculate the modular multiplicative inverse of e modulo z and assign it to d using the mod_inverse function.

Step 7: Input: Prompt the user to enter a message (an integer) to be encrypted. Ensure that the message is less than n and relatively prime to n.

Step 8: Check if the greatest common divisor (GCD) of the message and n is equal to 1.

If not, inform the user that the message must be relatively prime to n, and prompt them to enter another message.

Step 9: Check if the greatest common divisor (GCD) of the message and n is equal to 1.

If not, inform the user that the message must be relatively prime to n, and prompt them to enter another message.

Step 10: Decrypt the cipher text using the RSA decryption formula, decrypted_message = (cipher_text ** d) % n.

Step 11: Print the original message, cipher text, and decrypted message to demonstrate the RSA encryption and decryption process.

Program:

```
import random

import math


def gcd(a, b):

    if b == 0:

        return a

    return gcd(b, a % b)


def mod_inverse(a, m):
    m0, x0, x1 = m, 0, 1
    while a > 1:
        q = a // m
        m, a = a % m, m
        x0, x1 = x1 - q * x0, x0
    return x1 + m0 if x1 < 0 else x1


def encrypt(message, e, n):
    return (message ** e) % n


def decrypt(cipher, d, n):
    return (cipher ** d) % n


p = int(input("Enter a prime number p: "))
q = int(input("Enter another prime number q: "))


n = p * q


z = (p - 1) * (q - 1)
```

```
def is_prime(num):

    if num <= 1:

        return False

    if num == 2:

        return True

    if num % 2 == 0:

        return False

    for i in range(3, int(math.sqrt(num)) + 1, 2):

        if num % i == 0:

            return False

    return True


e = random.randint(2, z - 1)

while not is_prime(e):

    e = random.randint(2, z - 1)


d = mod_inverse(e, z)


message = int(input("Enter a message (a number, should be less than n and relatively prime to n): 
"))
```

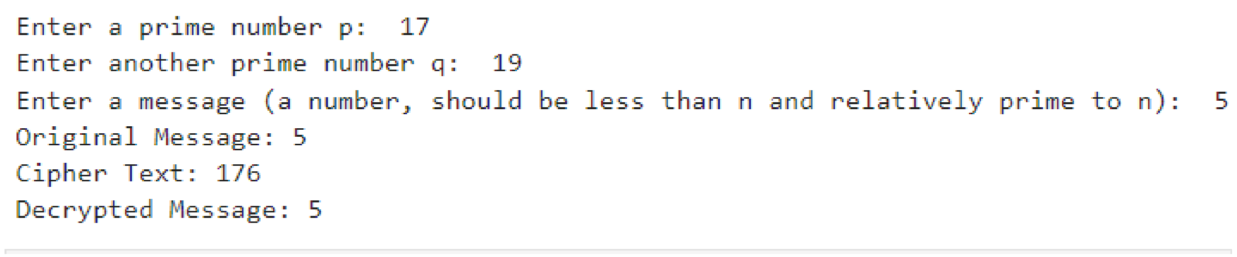
```
while gcd(message, n) != 1:
    print("The message must be relatively prime to n. Please choose another message.")
    message = int(input("Enter a message (a number, should be less than n and relatively prime to n): "))

cipher_text = encrypt(message, e, n)

decrypted_message = decrypt(cipher_text, d, n)

print(f"Original Message: {message}")
print(f"Cipher Text: {cipher_text}")
print(f"Decrypted Message: {decrypted_message}")
```

Output Screenshot:



```
Enter a prime number p:  17
Enter another prime number q:  19
Enter a message (a number, should be less than n and relatively prime to n):  5
Original Message: 5
Cipher Text: 176
Decrypted Message: 5
```

Result

Thus, the experiment to Implement Asymmetric Cipher (RSA) is carried out successfully and obtained the required output.