Experiment no. 6

Aim - Implementation & analysis of RSA cryptosystem & digital Signature scheme using RSA.

Theory -

RSA - RSA algorithm is asymmetric cryptography algorithm. Asymmetric actually means it works on two different keys i.e. Public key & private key. Since this is asymmetric, nobody else except browser can decrypt data even if a third party has public key of browser.

Working of RSA

It works on two keys

- Public key - It comprises of two numbers in which one number is the result of product of two large prime numbers. This key is provided to all its users.

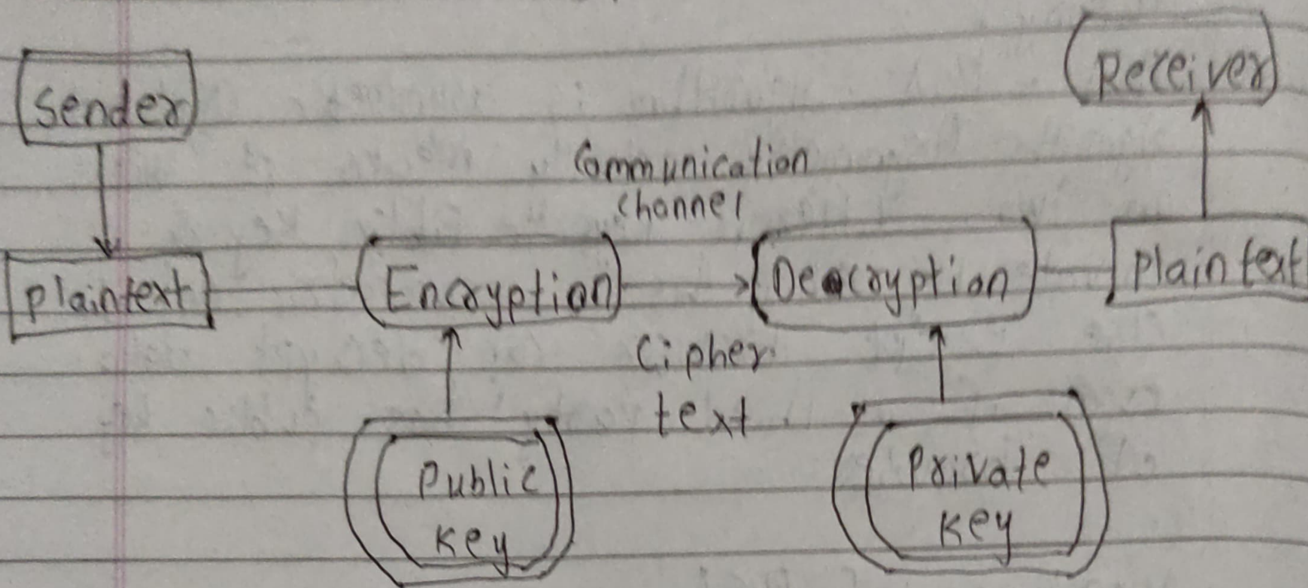
- Private key - It is derived from two prime numbers involved in public key & it remains always in private.

Characteristics :

① It is a public encryption technique.
② It is safe for data exchange over internet.
③ It maintains confidentality of data.
⑤ RSA has high toughness as breaking into

the keys by interceptions is very difficult.

RSA



Advantages of RSA

- It is very easy to implement RSA algorithm
- RSA algorithm is Safe & Secure For transmitting confidential data.
- Cracking RSA algorithm is very difficult as it involves complex mathematics.
- Sharing public key to users is easy.

Disadvanges of RSA

- It may fail sometimes because for complete encryption both symmetric & asymmetric encryption is required & RSA uses asymmetric encryption only.
- It requires 3rd party to verify reliability of public key sometimes, also slow downs data transfer

RSA

PROGRAM :

```
import random
def gcd(a, b):
    while b != 0:
        a, b = b, a % b
    return a


def multiplicative_inverse(e, phi):
    d = 0
    x1 = 0
    x2 = 1
    y1 = 1
    temp_phi = phi
    while e > 0:
        temp1 = temp_phi//e
        temp2 = temp_phi - temp1 * e
        temp_phi = e
        e = temp2
        x = x2 - temp1 * x1
        y = d - temp1 * y1
        x2 = x1
        x1 = x
        d = y1
```

```
        y1 = y
    if temp_phi == 1:
        return d + phi

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_key_pair(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 = (p-1) * (q-1)
    e = random.randrange(1, phi)
    g = gcd(e, phi)
    while g != 1:
```

```
        e = random.randrange(1, phi)
        g = gcd(e, phi)
    d = multiplicative_inverse(e, phi)
    return ((e, n), (d, n))


def encrypt(pk, plaintext):
    key, n = pk
    cipher = [pow(ord(char), key, n) for char in plaintext]
    return cipher


def decrypt(pk, ciphertext):
    key, n = pk
    aux = [str(pow(char, key, n)) for char in ciphertext]
    plain = [chr(int(char2)) for char2 in aux]
    return ''.join(plain)
if __name__ == '__main__':
    p = int(input("Enter a prime number (17, 19, 23, etc): "))
    q = int(input("Enter another prime number (Not one you entered
above): "))
    print("Generating your public / private key-pairs now . . .")
    public, private = generate_key_pair(p, q)
    print("Your public key is ", public, " and your private key is ",
private)
    message = input("Enter a message to encrypt with your public key:
")
```

```
    encrypted_msg = encrypt(public, message)
    print("Your encrypted message is: ", ''.join(map(lambda x: str(x), encrypted_msg)))
    print("Decrypting message with private key ", private, " . . .")
    print("Your message is: ", decrypt(private, encrypted_msg))
```

OUTPUT :

```
C:\Windows\System32\cmd.exe

D:\>python rsa.py
Enter a prime number (17, 19, 23, etc): 7
Enter another prime number (Not one you entered above): 11
Generating your public / private key-pairs now . . .
Your public key is  (29, 77)  and your private key is  (89, 77)
Enter a message to encrypt with your public key: HELLO
Your encrypted message is:  4648767639
Decrypting message with private key  (89, 77)  . . .
Your message is:  HELL☻

D:\>
```

DIGITAL SIGNATURE USING RSA

STEP 1

STEP 2

Hash output(hex):

a94a8fe5ccb19ba61c4c0873d391e987982fbbc

Input to RSA(hex):

a94a8fe5ccb19ba61c4c0873d391e987982fbbc Apply RSA

STEP 3

RSA public key

Public exponent (hex, F4=0x10001):

10001

Modulus (hex):

a5261939975948bb7a58dffe5ff54e65f0498f9175f5a09288810b8975871e99
af3b5dd94057b0fc07535f5f97444504fa35169d461d0d30cf0192e307727c06
5168c788771c561a9400fb49175e9e6aa4e23fe11af69e9412dd23b0cb6684c4
c2429bce139e848ab26d0829073351f4acd36074eafd036a5eb83359d2a698d3

1024 bit | 1024 bit (e=3) | 512 bit | 512 bit (e=3)

STEP 4

Input to RSA(hex):

a94a8fe5ccb19ba61c4c0873d391e987982fbbc Apply RSA

Digital Signature(hex):

63c3e5c965d89d7b0192e478fe3f4dd52113dd08bd067f2c7c029a4e2f8f885b
6e0d0b10d524cbc01fd3f1a7b3a1e0f3724b0c87fe5ace2fe32df06144748eb9
026d0f8707148d42f601a0043d70a7d0b72148bb616a5cc36368aa012fafdf98
67ec00100bc2f4952d4447e7ca4c1a35648f4f9c0d680e037d4ec31d1d62fff1

Digital Signature(base64):

Y8PlyWXYnXsBkuR4/j9N1SET3Qi9Bn8sfAKaTi+PiFtuDQsQ1STLwB/T8aezoeDz
cksMh/5azi/jLfBhRHSOuQJtD4cHFI1C9gGgBD1wp9C3IUi7YWpcw2NoqgEvr9+Y
Z+wAEAvC9JUtREfnykwaNWSPT5wNaA4DfU7DHR1i//E=

Status:

Time: 13ms

CONCLUSION : Hence we can conclude that we have learned and implemented analysis of RSA cryptosystem and digital signature scheme using RSA