Cryptography & Network Security Lab

PRN/ Roll No: 2019BTECS00090

Full name: Udaykumar Gadikar

Assignment No. 12

Title: RSA Algorithm

Aim: To Demonstrate RSA Algorithm

Theory:

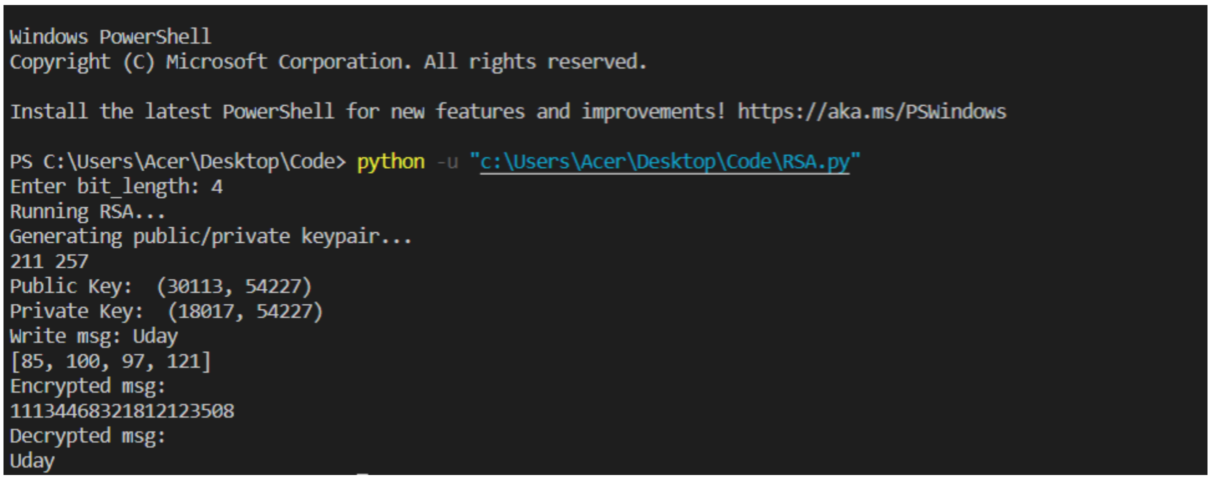
RSA (Rivest–Shamir–Adleman) is a public-key cryptosystem that is widely used for secure data transmission.

An RSA user creates and publishes a public key based on two large prime numbers, along with an auxiliary value. The prime numbers are kept secret. Messages can be encrypted by anyone, via the public key, but can only be decoded by someone who knows the prime numbers.

Code:

```python
from math import sqrt
import random
from random import randint as rand


def gcd(a, b):
    if b == 0:
        return a
    else:
        return gcd(b, a % b)


def mod_inverse(a, m):
    for x in range(1, m):
        if (a * x) % m == 1:
            return x
    return -1


def isprime(n):
    if n < 2:
        return False
    elif n == 2:
        return True
    else:
        for i in range(2, int(sqrt(n)) + 1, 2):
            if n % i == 0:
                return False
    return True


p = rand(1, 1000)
q = rand(1, 1000)


def generate_keypair(p, q, keysize):
    nMin = 1 << (keysize - 1)
    nMax = (1 << keysize) - 1
    primes = [2]

    start = 1 << (keysize // 2 - 1)
    stop = 1 << (keysize // 2 + 1)

    if start >= stop:
        return []

    for i in range(3, stop + 1, 2):
        for p in primes:
            if i % p == 0:
                break
        else:
            primes.append(i)

    while (primes and primes[0] < start):
        del primes[0]

    while primes:
        p = random.choice(primes)
        primes.remove(p)
        q_values = [q for q in primes if nMin <= p * q <= nMax]
        if q_values:
            q = random.choice(q_values)
            break
    print(p, q)
    n = p * q
    phi = (p - 1) * (q - 1)

    e = random.randrange(1, phi)
    g = gcd(e, phi)

    while True:
        e = random.randrange(1, phi)
        g = gcd(e, phi)
        d = mod_inverse(e, phi)
        if g == 1 and e != d:
            break

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


def encrypt(msg_plaintext, package):
    e, n = package
    msg_ciphertext = [pow(ord(c), e, n) for c in msg_plaintext]
    return msg_ciphertext


def decrypt(msg_ciphertext, package):
    d, n = package
    msg_plaintext = [chr(pow(c, d, n)) for c in msg_ciphertext]
    return (''.join(msg_plaintext))


if __name__ == "__main__":
    bit_length = int(input("Enter bit_length: "))
    print("Running RSA...")
    print("Generating public/private keypair...")
    public, private = generate_keypair(p, q, 2**bit_length)
    print("Public Key: ", public)
    print("Private Key: ", private)
    msg = input("Write msg: ")
    print([ord(c) for c in msg])
    encrypted_msg = encrypt(msg, public)
    print("Encrypted msg: ")
    print(''.join(map(lambda x: str(x), encrypted_msg)))
    print("Decrypted msg: ")
    print(decrypt(encrypted_msg, private))
```

Output:



```
Windows PowerShell
Copyright (C) Microsoft Corporation. All rights reserved.

Install the latest PowerShell for new features and improvements! https://aka.ms/PSWindows

PS C:\Users\Acer\Desktop\Code> python -u "c:\Users\Acer\Desktop\Code\RSA.py"
Enter bit_length: 4
Running RSA...
Generating public/private keypair...
211 257
Public Key:  (30113, 54227)
Private Key:  (18017, 54227)
Write msg: Uday
[85, 100, 97, 121]
Encrypted msg:
11134468321812123508
Decrypted msg:
Uday
```

<u>Conclusion</u>:

The security of RSA relies on the practical difficulty of factoring the product of two large prime numbers, the "factoring problem".