

PRACTICAL: 5

AIM:

RSA algorithm is a public key encryption technology and is considered the most secure way of encryption to secure sensitive data, particularly when it is being sent over an insecure network such as the internet. The public and private key generation algorithm is the most complex part of the RSA algorithm. The strength of RSA is the difficulty of factoring large integers that are the product of two large prime numbers, which is considered infeasible due to the time it would take using even today's highly configured computers. Implement the RSA algorithm.

THEORY:

The RSA algorithm is a public-key signature algorithm developed by Ron Rivest, Adi Shamir, and Leonard Adleman. Their paper was first published in 1977, and the algorithm uses logarithmic functions to keep the working complex enough to withstand brute force and streamlined enough to be fast post-deployment. The image below shows it verifies the digital signatures using RSA methodology.

RSA can also encrypt and decrypt general information to securely exchange data along with handling digital signature verification. The image above shows the entire procedure of the RSA algorithm. You will understand more about it in the next section.

RSA in Data Encryption

When using RSA for encryption and decryption of general data, it reverses the key set usage. Unlike signature verification, it uses the receiver's public key to encrypt the data, and it uses the receiver's private key in decrypting the data. Thus, there is no need to exchange any keys in this scenario.

There are two broad components when it comes to RSA cryptography, they are:

- **Key Generation:** Generating the keys to be used for encrypting and decrypting the data to be exchanged.
- **Encryption/Decryption Function:** The steps that need to be run when scrambling and recovering the data.

Steps in RSA Algorithm

Keeping the image above in mind, go ahead and see how the entire process works, starting from creating the key pair, to encrypting and decrypting the information.

Key Generation

- You need to generate public and private keys before running the functions to generate your ciphertext and plaintext. They use certain variables and parameters, all of which are explained below:
- Choose two large prime numbers (p and q)
- Calculate $n = p * q$ and $z = (p-1)(q-1)$
- Choose a number e where $1 < e < z$
- Calculate $d = e^{-1} \bmod (p-1)(q-1)$
- You can bundle private key pair as (n, d)
- You can bundle public key pair as (n, e)

Encryption/Decryption Function

Once you generate the keys, you pass the parameters to the functions that calculate your ciphertext and plaintext using the respective key.

- If the plaintext is m , ciphertext = $m^e \bmod n$.
- If the ciphertext is c , plaintext = $c^d \bmod n$

To understand the above steps better, you can take an example where $p = 17$ and $q = 13$. Value of e can be 5 as it satisfies the condition $1 < e < (p-1)(q-1)$.

$$N = p * q = 221$$

$$D = e^{-1} \bmod (p-1)(q-1) = 29$$

Public Key pair = (221,5)

Private Key pair = (221,29)

If the plaintext(m) value is 10, you can encrypt it using the formula $me \bmod n = 82$.

To decrypt this ciphertext(c) back to original data, you must use the formula $cd \bmod n = 29$.

You can now look at the factors that make the RSA algorithm stand out versus its competitors in the advantages section.

Advantages of RSA

- **No Key Sharing:** RSA encryption depends on using the receiver's public key, so you don't have to share any secret key to receive messages from others.
- **Proof of Authenticity:** Since the key pairs are related to each other, a receiver can't intercept the message since they won't have the correct private key to decrypt the information.
- **Faster Encryption:** The encryption process is faster than that of the DSA algorithm.
- **Data Can't Be Modified:** Data will be tamper-proof in transit since meddling with the data will alter the usage of the keys. And the private key won't be able to decrypt the information, hence alerting the receiver of manipulation.

CODE:

```
import sympy
import random
import math

p = sympy.randprime(999999, 99999999)
q = sympy.randprime(999999, 99999999)
if(q == p):
    q = sympy.randprime(999999, 99999999)
n = p*q
z = (p-1)*(q-1) # phi(n)

## TODO Key generation Part

e = int(random.randint(2,z-1)) # public key
while(math.gcd(e,z) != 1):
    e = int(random.randint(2,z-1))

k = 1
while((1+k*z)%e != 0):
    k+=1
d = (1+k*z)/e

# print(d)

def encryption(value):
    result = (value**e)%n
    return result

def decryption(value):
    result = (value**d)%n
    return result
```

```
def ciphertextCharwise(m):
    cipher = ""
    for ch in m:
        cipher = cipher + chr((ord(ch) ** e) % n)
    return cipher

def decryptCipherCharwise(cipher):
    print(cipher)
    plain = ""
    for letter in cipher:
        plain = plain + chr( (int(cipher) ** d) % n)
    return plain

def char_by_char(m):
    cipherText = ciphertextCharwise(m)
    plainText = decryptCipherCharwise(cipherText)
    return plainText

def block_wise(block_size,m):
    size = len(m)
    while(size % block_size != 0):
        m = m + 'z'

    encryptedValue = []
    decryptedValue = []

    # block wise splitting here
    i = 0
    while(i <= size):
        cipher = ciphertextCharwise(m[i:i+block_size])
        encryptedValue.append(cipher)
        decryptedValue.append(decryptCipherCharwise(cipher))
        i += block_size
    return decryptedValue

def isBlockSize_Valid (block_size,m):
    text_size = len(m)
    if(block_size <= 1 or block_size >= text_size):
        return False
    return True

def main():
    m = input("Enter the plain text: ")

    print("\nEnter 1 if u want to encrypt character by character")
    print("Enter 2 if u want to encrypt your message block wise")
    option = int(input("\nEnter here :"))

    if(option == 1):
        decryptedValue = char_by_char(m)
```

```
else:
    block_size = int(input("\nEnter the size of block: "))
    while(isBlockSize_Valid(block_size,m) == False):
        block_size = int(input("\nEnter the size of block: "))
    decryptedValues = block_wise(block_size,m)
    print(decryptedValues)
```

```
main()
```

OUTPUT:

PROBLEMS OUTPUT **TERMINAL** DEBUG CONSOLE

Python + - [] [X] ^ X

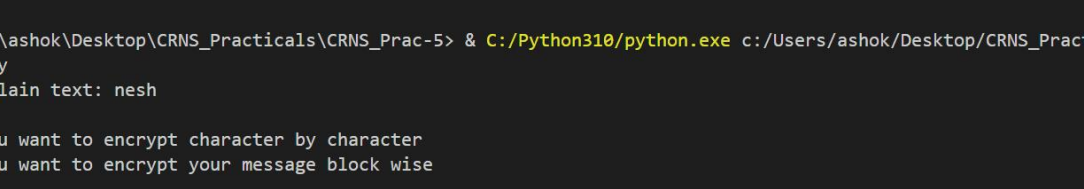
```
PS C:\Users\ashok\Desktop\CRNS_Practicals\CRNS_Prac-5> & C:/Python310/python.exe c:/Users/ashok/Desktop/CRNS_Practicals/CRNS_Prac-5/RSA.py
Enter the plain text: nesh

Enter 1 if u want to encrypt character by character
Enter 2 if u want to encrypt your message block wise

Enter here :2

Enter the size of block: 2
%e
Traceback (most recent call last):
  File "c:\Users\ashok\Desktop\CRNS_Practicals\CRNS_Prac-5\RSA.py", line 94, in <module>
    main()
  File "c:\Users\ashok\Desktop\CRNS_Practicals\CRNS_Prac-5\RSA.py", line 91, in main
    decryptedValues = block_wise(block_size,m)
  File "c:\Users\ashok\Desktop\CRNS_Practicals\CRNS_Prac-5\RSA.py", line 68, in block_wise
    decryptedValue.append(decryptCipherCharwise(cipher))
  File "c:\Users\ashok\Desktop\CRNS_Practicals\CRNS_Prac-5\RSA.py", line 46, in decryptCipherCharwise
    plain = plain + chr( (int(cipher) ** d) % n)
ValueError: invalid literal for int() with base 10: '%e'
PS C:\Users\ashok\Desktop\CRNS_Practicals\CRNS_Prac-5> |
```

Selecting option 2 blockwise cipher



```
PS C:\Users\ashok\Desktop\CRNS_Practicals\CRNS_Prac-5> & C:/Python310/python.exe c:/Users/ashok/Desktop/CRNS_Practicals/CRNS_Prac-5/RSA.py
Enter the plain text: nesh

Enter 1 if u want to encrypt character by character
Enter 2 if u want to encrypt your message block wise

Enter here :2

Enter the size of block: 4

Enter the size of block: 5

Enter the size of block: 0

Enter the size of block: 1

Enter the size of block: 2
φe
```

Entering an invalid block size in block cipher

```

PS C:\Users\ashok\Desktop\CRNS_Practicals\CRNS_Prac-5> & C:/Python310/python.exe c:/Users/ashok/Desktop/CRNS_Practicals/CRNS_Prac-5/RSA.py
Enter the plain text: nesh

Enter 1 if u want to encrypt character by character
Enter 2 if u want to encrypt your message block wise

Enter here :1
^bu
Traceback (most recent call last):
  File "c:\Users\ashok\Desktop\CRNS_Practicals\CRNS_Prac-5\RSA.py", line 94, in <module>
    main()
  File "c:\Users\ashok\Desktop\CRNS_Practicals\CRNS_Prac-5\RSA.py", line 86, in main
    decryptedValue = char_by_char(m)
  File "c:\Users\ashok\Desktop\CRNS_Practicals\CRNS_Prac-5\RSA.py", line 51, in char_by_char
    plainText = decryptCipherCharwise(cipherText)
  File "c:\Users\ashok\Desktop\CRNS_Practicals\CRNS_Prac-5\RSA.py", line 46, in decryptCipherCharwise
    plain = plain + chr( (int(cipher) ** d) % n)
ValueError: invalid literal for int() with base 10: '\x91^bu'
PS C:\Users\ashok\Desktop\CRNS_Practicals\CRNS_Prac-5>

```

Selecting option 1 for stream cipher

LEARNING OUTCOME:

Implementing RSA algorithm to do encryption and decryption on given text to ensure security principles i.e. confidentiality,integrity and availability.

REFERENCES:

- <https://www.geeksforgeeks.org/rsa-algorithm-cryptography/>
- <https://www.cs.toronto.edu/~david/course-notes/csc110-111/07-cryptography/05-rsa-cryptosystem-implementation.html>

