Cryptography and Network Security Lab

Assignment 10
Implementation and Understanding of RSA Algorithm

2019BTECS00058
Devang K

Batch: B2

Title: Implementation and Understanding of RSA Algorithm

Aim: To Study, Implement and Demonstrate the RSA Algorithm

Theory:

RSA (Rivest–Shamir–Adleman) is a public-key cryptosystem that is widely used for secure data transmission. It is also one of the oldest. The acronym "RSA" comes from the surnames of Ron Rivest, Adi Shamir and Leonard Adleman, who publicly described the algorithm in 1977. An equivalent system was developed secretly in 1973 at GCHQ (the British signals intelligence agency) by the English mathematician Clifford Cocks. That system was declassified in 1997.

In a public-key cryptosystem, the encryption key is public and distinct from the decryption key, which is kept secret (private). 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.

The security of RSA relies on the practical difficulty of factoring the product of two large prime numbers, the "factoring problem". Breaking RSA encryption is known as the RSA problem. Whether it is as difficult as the factoring problem is an open question. There are no published methods to defeat the system if a large enough key is used.

RSA is a relatively slow algorithm. Because of this, it is not commonly used to directly encrypt user data. More often, RSA is used to transmit shared keys for symmetric-key cryptography, which are then used for bulk encryption–decryption.

Operation:

The RSA algorithm involves four steps: key generation, key distribution, encryption, and decryption.

A basic principle behind RSA is the observation that it is practical to find three very large positive integers e, d, and n, such that with modular exponentiation for all integers m (with $0 \leq m < n$):

$$(m^e)^d \equiv m \pmod{n}$$

and that knowing e and n, or even m, it can be extremely difficult to find d. The triple bar (≡) here denotes modular congruence (which is to say that when you divide (me)d by n and m by n, they both have the same remainder).

In addition, for some operations it is convenient that the order of the two exponentiations can be changed and that this relation also implies:

$$(m^d)^e \equiv m \pmod{n}.$$

RSA involves a public key and a private key. The public key can be known by everyone and is used for encrypting messages. The intention is that messages encrypted with the public key can only be decrypted in a reasonable amount of time by using the private key. The public key is represented by the integers n and e, and the private key by the integer d (although n is also used during the decryption process, so it might be considered to be a part of the private key too). m represents the message (previously prepared with a certain technique explained below).

This is the crux of the RSA algorithm.

Essentially, we first convert the string to numbers, then encrypt the large number. Decryption is similar – we get our original number which can be converted to the plaintext.

Code:

```
from os import system, name
import msvcrt
import rsa
import sys

def ConvertToInt(message):
    theNumber = "1"
    for m in message:
        theASCII = str(ord(m))
        if len(theASCII) == 1:
            theASCII = "00"+theASCII
        elif len(theASCII) == 2:
            theASCII = "0"+theASCII
        theNumber += theASCII
    return int(theNumber)

def clear():
    if name == 'nt':
        _ = system('cls')
    else:
        _ = system('clear')

def Generate_Keys():
    clear()
    print("\n\n*************************Key
Generation*************************\n")
    (pubkey, privkey) = rsa.newkeys(512)
    print("Your Public Keys 'e' and 'n' respectively are:")
    print(pubkey.e)
    print()
    print(pubkey.n)
    print("\nYour Private Keys 'd' and 'n' respectively are:")
    print(privkey.d)
    print()
    print(privkey.n)
    print("Warning: Don't share your Private Keys with Anyone!")
    print("Press Any Key to Go Back")
    msvcrt.getch()
    home()

def Encrypt_Dat():
    clear()
    print("\n\n*************************Encryption*************************\n"
)
    print("Enter the Message to Encrypt:")
    contents = []
    while True:
```

```python
        try:
            line = input()
            line.lstrip()
            line.rstrip()
            if len(line) == 0:
                break
        except EOFError:
            break
        contents.append(line)
    print("Enter 'n' of Receiver:")
    n = input()
    print("Enter 'e' of Receiver:")
    e = input()
    encry = []
    print("\nEncoded Message is:")
    for line in contents:
        mess = ConvertToInt(line)
        mess = pow(mess, int(e), int(n))
        encry.append(mess)
    for line in encry:
        print(line)

    print("\nPress Any Key to Continue.")
    msvcrt.getch()
    home()

def PowMod(a, n, mod):
    if n == 0:
        return 1 % mod
    elif n == 1:
        return a % mod
    else:
        b = PowMod(a, n // 2, mod)
        b = b * b % mod
        if n % 2 == 0:
          return b
        else:
          return b * a % mod


def ExtendedEuclid(a, b):
    if b == 0:
        return (1, 0)
    (x, y) = ExtendedEuclid(b, a % b)
    k = a // b
    return (y, x - k * y)

def InvertModulo(a, n):
    (b, x) = ExtendedEuclid(a, n)
```

```python
    if b < 0:
        b = (b % n + n) % n
    return b

def Decrypt(ciphertext, d, n):
  return ConvertToStr(PowMod(ciphertext, d, n))

def ConvertToStr(numbers):
    numbers = str(numbers)
    numbers = numbers[1:]
    theMessageList = [numbers[i:i+3] for i in range(0, len(numbers), 3)]
    theMessage = ""
    for tml in theMessageList:
        theMessage += chr(int(tml))
    return theMessage

def Decrypt_Dat():
    clear()
    print("\n\n*************************Decryption*************************\n"
)
    print("Enter Message to Decrypt:")
    obey = []
    while True:
        try:
            line = input()
            line.lstrip()
            line.rstrip()
            if len(line) == 0:
                break
        except EOFError:
            break
        obey.append(int(line))

    # print("\nEnter Private Key 'p':")
    # p = int(input())
    # print("\nEnter Private Key 'q':")
    # q = int(input())
    print("\nEnter Private Key 'd':")
    d = int(input())
    print("\nEnter Your Public Key 'n':")
    n = int(input())

    print("\n\nMessage as Decrypted:")
    for line in obey:
        print(Decrypt(line, d, n))

    print("\nPress Any Key to Continue.")
    msvcrt.getch()
```

```
    home()

def home():
    clear()
    print("\n\n*************************DeSipher*************************\n")
    print("Choose your Action:")
    print("\t1. Encrypt Data.")
    print("\t2. Decrypt Data.")
    print("\t3. Generate Public and Private Keys.")
    print("\t4. Exit.")
    print("\nYour Choice: ", end='')
    inp = input()
    if inp == '1':
        Encrypt_Dat()
    elif inp == '2':
        Decrypt_Dat()
    elif inp == '3':
        Generate_Keys()
    elif inp == '4':
        sys.exit()
    else:
        print("Invalid Choice.")
        print("Try Again.")
        print("Press Any Key To Continue.")
        msvcrt.getch()
        home()

home()
```

We now illustrate with examples:

We need to first generate 2 public and private keys:

For our 2 users – Alice and Bob.

```
************************Key Generation************************

Your Public Keys 'e' and 'n' respectively are:
65537

8704381402772414233929206680194105759156734812089972546851983094678461015273
2575304465688320643283854369546021351510922996618745735634343468366461432006
19

Your Private Keys 'd' and 'n' respectively are:
6215802518420876545278039468286374865015719199929974139687089870316797765917
9841807715160369446897354632364937297852227168991886385967185257248613249278
73

8704381402772414233929206680194105759156734812089972546851983094678461015273
2575304465688320643283854369546021351510922996618745735634343468366461432006
19
Warning: Don't share your Private Keys with Anyone!
Press Any Key to Go Back
```

Your Public Keys 'e' and 'n' respectively are: 65537 87043814027724142339292066801941057591567348120899725468519830946784610152732575304465688320643283854369546021351510922996618745735634343468366461432006 19 Your Private Keys 'd' and 'n' respectively are: 62158025184208765452780394682863748650157191999299741396870898703167977659179841807715160369446897354632364937297852227168991886385967185257248613249278 73 87043814027724142339292066801941057591567348120899725468519830946784610152732575304465688320643283854369546021351510922996618745735634343468366461432006 19

Then for Bob:

```
*************************Key Generation*************************

Your Public Keys 'e' and 'n' respectively are:
65537

9492772589453858884710048085093768899664893814818005982569531802137122059051
2835282892841937375981766006663182494394973121358036862395922692767173864933
39

Your Private Keys 'd' and 'n' respectively are:
5053387401023333823180251729446746255878493300141455860351951195705630623878
7296857827760814541523114198257078593904452749063551671732918371172127865004
17

9492772589453858884710048085093768899664893814818005982569531802137122059051
2835282892841937375981766006663182494394973121358036862395922692767173864933
39
Warning: Don't share your Private Keys with Anyone!
Press Any Key to Go Back
```

Your Public Keys 'e' and 'n' respectively are:

65537

9492772589453858884710048085093768899664893814818005982569531802137122059051283528289284193737598176600666318249439497312135803686239592269276717386493339

Your Private Keys 'd' and 'n' respectively are:

505338740102333382318025172944674625587849330014145586035195119570563062387872968578277608145415231141982570785939044527490635516717329183711721278650041 7

9492772589453858884710048085093768899664893814818005982569531802137122059051283528289284193737598176600666318249439497312135803686239592269276717386493339

Now, say Alice wishes to share a message to Bob. She would send the message using Bob’s public key.

She wishes to send – ‘Osama is spotted in Abbottabad. Reach ASAP’.

This would work as follows:

```
*************************Encryption*************************

Enter the Message to Encrypt:
Osama is spotted in Abbottabad. Reach ASAP

Enter 'n' of Receiver:
94927725894538588847100480850937688996648938148180059825695318021371220590512835282892841937375981766006663182494394973121358036862395922692767173864933
39
Enter 'e' of Receiver:
65537

Encoded Message is:
934536386169434822155074140039413961590115349411303871581406421573978810690740833846352170570471030154443932822417727809621922487455472926621759649757666
6

Press Any Key to Continue.
```

Encoded Message is:

934536386169434822155074140039413961590115349411303871581406421573978810690740833846352170570471030154443932822417727809621922487455472926621759649757666

When Bob would receive this message, he would decrypt using his private key.

```
*************************Decryption*************************

Enter Message to Decrypt:
934536386169434822155074140039413961590115349411303871581406421573978810690740833846352170570471030
154443932822417727809621922487455472926621759649757666


Enter Private Key 'd':
505338740102333382318025172944674625587849330014145586035195119570563062387872968578277608145415231
1419825707859390445274906355167173291837117212786500417

Enter Your Public Key 'n':
949277258945385888471004808509376889966489381481800598256953180213712205905128352828928419373759817
6600666318249439497312135803686239592269276717386493339


Message as Decrypted:
Osama is spotted in Abbottabad. Reach ASAP

Press Any Key to Continue.
```

We receive our plaintext back.

Now say Bob wishes to message ‘Okay’ to Alice.

```
*************************Encryption*************************

Enter the Message to Encrypt:
Okay

Enter 'n' of Receiver:
870438140277241423392920668019410575915673481208997254685198309467846101527325753044656883206432838
5436954602135151092299661874573563434346836646143200619
Enter 'e' of Receiver:
65537

Encoded Message is:
267593717524169236984300247477434217565292212580899954857233723530423919847520766521489841401500561
1829141650046814145233425278378678585515868427620613517

Press Any Key to Continue.
```

Encoded Message is:

2675937175241692369843002474774342175652922125808999548572337235
3042391984752076652148984140150056118291416500468141452334252783
78678585515868427620613517

Then Alice would decrypt using her private key:

```
************************Decryption************************

Enter Message to Decrypt:
26759371752416923698430024747743421756529221258089995485723372353042391984752076652148984140150056118291416500468141452334252783786785855158684276206135 17
```

```
Enter Private Key 'd':
6215802518420876545278039468286374865015719199929974139687089870316797765917984180771516036944689735463236493729785222716899188638596718525724861324927873

Enter Your Public Key 'n':
87043814027724142339292066801941057591567348120899725468519830946784610152732575304465688320643283854369546021351510922996618745735634343468366461432006 19


Message as Decrypted:
Okay

Press Any Key to Continue.
```

Thus, she received the message from Bob.

Thu, we illustrated working of RSA in the code.

Conclusion:

Thus, the RSA algorithm was studied and demonstrated with the code.