

Assignment 3 Analysis

For this assignment you are required to use 64-bit integers to store and do the underlying basic arithmetic operations. If you are using python programming language then you must use `numpy.uint64` and `numpy.int64` for storing unsigned and signed integers respectively.

1. Write programs that take as input 3 n-bit numbers x , y and N and computes the following.
 - a. $(x + y) \bmod N$ (10)
 - b. $x \cdot y \bmod N$ (10)

Note that n is typically much larger than 64.
2. Write programs to implement the following for n-bit numbers.
 - a. Euclidean algorithm (10)
 - b. Extended euclidean algorithm (10)
 - c. Multiplicative inverse algorithm (10)
 - d. Square and Multiply algorithm (10)
3. Write a program that computes the Jacobi symbol (a/N) . (10)
4. Implement the Solovay-Strassen and the Miller-Rabin algorithm for primality testing. (20 + 20 = 40)
5. Implement the plain RSA encryption scheme. (40)

Note:

-All codes have been written in Python and executed on Anaconda Spyder(Python-3.7).

-python files can be directly executed by using below commands in terminal

->*python que1.py*

//change file name according to question number.

Details of Question 1

1(a) $(x+y) \bmod N$

1(b) $(x.y) \bmod N$

3 n-bit numbers-- x, y and N

n = 64-bit

Functions:

modPlus - to calculate $(x+y) \bmod N$

modMult - to calculate $(x*y) \bmod N$

Link to file - <https://github.com/zeya2u9/Applied-Cryptography/blob/main/Assignment-3/que1.py>

Method: modPlus

1. Taking mod N of both x and y separately.

$$(x+y) \bmod N = ((x \bmod N) + (y \bmod N)) \bmod N$$

$$r1 = (x \% N)$$

$$r2 = (y \% N)$$

2. Checking if $r1+r2$ exceeds the maximum possible value(overflow error).

If It does not –

Simply add both and take mod N

Else

$$\text{Temp} = N - r1$$

$$\text{Result} = r2 - \text{Temp}$$

Main idea here is to take out the difference value from r2 to make r1 a multiple of N.

Method: modMult

1. Taking mod N of both x and y separately.

$$(x*y) \bmod N = ((x \bmod N) * (y \bmod N)) \bmod N$$

$$r1 = (x \% N)$$

$$r2 = (y \% N)$$

2. Checking if $r1*r2$ exceeds the maximum possible value(overflow error).

If It does not –

Simply multiply both and take mod N

Else

➤ Now it can be solved by using modPlus of $r1 \bmod N$ by $r2$ number of times.(worst case scenario)

➤ Another efficient method is to keep dividing the $r2$ until it becomes 1 or

$(r1*r2) < \text{maximum_possible_value}$. And keep updating $r1$ as $r1 = (r1+r1) \bmod N$

In case $r2$ is odd – take a variable as **extra** which will store the extra $r1$ as $\text{extra} = (\text{extra}+r1) \% N$

In case $r2$ is even – do not update **extra** variable

➤ In the end, $\text{result} = (r1*r2 + \text{extra}) \bmod N$

Output:

■ Anaconda Powershell Prompt

```
(base) PS D:\Research 1.0\1 Applied Cryptography\Assignment 3> python que1.py
Constraint:: only upto 64-bit numbers are allowed
Some examples::
x= 18446744073709551614 y= 18446744073709551614 N= 18446744073709551615
(x+y)mod N = 18446744073709551613
x= 18446744073709551614 y= 18446744073709551614 N= 18446744073709551615
modMult done! Time elapsed: 0.0 seconds
x= 18446744073709551614 y= 18446744073709551614 N= 18446744073709551615
(x.y)mod N = 1
Enter x,y and N(keep hitting Enter key after each input):
11111111
22222222
33333333
x= 11111111 y= 22222222 N= 33333333
x= 11111111 y= 22222222 N= 33333333
(x+y)mod N = 33333333
(x.y)mod N = 24222222
(base) PS D:\Research 1.0\1 Applied Cryptography\Assignment 3>
```

Details of Question 2

- (a) Euclidean Algorithm **euclideanAlgo(x,y)**
- (b) Extended Euclidean algorithm **extEuclideanAlgo(x,y)**
- (c) Multiplicative inverse algorithm **multInverse(x,N)**
- (d) Square and Multiply algorithm **squAndmult(a,p,N)**

Link to code- <https://github.com/zeya2u9/Applied-Cryptography/blob/main/Assignment-3/que2.py>

(a)Euclidean Algorithm:

def euclideanAlgo(x,y):

 gcd =1 #initializing gcd

 r = 1 #initializing variable remainder as r

 if(x>y):

 a,b = x,y #fixing a as the bigger value

 else:

 b,a = x,y

 print(a,b)

 while(1):#keep diving a by b until the remainder is zero

 r = a%b #taking remainder in r

 if r==0: #checking if further division is needed

 gcd = b

 break

 a = b #keep updating a by b and b by the remainder r

 b = r

 return gcd

(b) Extended Euclidean algorithm:

The extended Euclidean algorithm is an extension to the Euclidean algorithm, and computes, in addition to the greatest common divisor (gcd) of integers a and b , also the coefficients of Bézout's identity, which are integers x and y such that $ax + by = \text{gcd}(a,b)$

It gives gcd and coefficients as output.

Method: Euclidean algorithm is explained above with the help of comments.

The extended Euclidean algorithm proceeds similarly, but adds two other sequences, as follows

$$\begin{array}{ll}
 r_0 = a & r_1 = b \\
 s_0 = 1 & s_1 = 0 \\
 t_0 = 0 & t_1 = 1 \\
 \vdots & \vdots \\
 r_{i+1} = r_{i-1} - q_i r_i & \text{and } 0 \leq r_{i+1} < |r_i| \quad (\text{this defines } q_i) \\
 s_{i+1} = s_{i-1} - q_i s_i \\
 t_{i+1} = t_{i-1} - q_i t_i \\
 \vdots &
 \end{array}$$

The computation also stops when $r_{k+1} = 0$ and gives

- r_k is the greatest common divisor of the input $a = r_0$ and $b = r_1$.
- The Bézout coefficients are s_k and t_k , that is $\gcd(a, b) = r_k = as_k + bt_k$

(c) Multiplicative inverse algorithm: It takes two numbers x, n as input and gives the multiplicative inverse as output. Output = $x^{-1} \bmod n$

> The idea is to use Extended Euclidean algorithms that takes two integers 'a' and 'b', finds their gcd and also find 'x' and 'y' such that

$$ax + by = \gcd(a, b)$$

To find multiplicative inverse of 'a' under 'm', we put $b = m$ in above formula. Since we know that a and m are relatively prime, we can put value of gcd as 1.

$$ax + my = 1$$

If we take modulo m on both sides, we get

$$ax + my \cong 1 \pmod{m}$$

We can remove the second term on left side as 'my (mod m)' would always be 0 for an integer y.

$$ax \cong 1 \pmod{m}$$

So the 'x' that we can find using Extended Euclid Algorithm is the multiplicative inverse of 'a'

Thus multInverse() method uses extEuclideanAlgo(), euclideanAlgo() and modPlus() methods internally to calculate gcd and coefficients.

(d) Square and Multiply algorithm : It takes numbers a,p and N as input. And gives $a^p \bmod N$ as output. Main idea of algorithm is explained as below from wikipedia:


The method is based on the observation that, for a positive integer n , we have

$$x^n = \begin{cases} x (x^2)^{\frac{n-1}{2}}, & \text{if } n \text{ is odd} \\
 (x^2)^{\frac{n}{2}}, & \text{if } n \text{ is even.} \end{cases}$$

This method uses the bits of the exponent to determine which powers are computed.

This method uses **modPlus()** and **modMult()** methods internally.

Output:

 Anaconda Powershell Prompt

```
(base) PS D:\Research 1.0\1 Applied Cryptography\Assignment 3> python que2.py
Constraint:: only upto 64-bit numbers are allowed
Some examples::
18446744073709551614 9223372036854775807
x= 18446744073709551614 y= 9223372036854775807
GCD(x,y) = 9223372036854775807
x= 1432 y= 123211
GCD(x,y) = 1
Bézout coefficients: -22973 267
26 15
x= 15 n= 26
Multiplicative Inverse( 0 - indicates Inverse does not exist) = 7
670000 9 68000

a= 670000 p= 9 N 68000
a^p mod N = 24000
Enter x,y and N(keep hitting Enter key after each input):
233
991
1991
991 233
1991 233
233 991 1991

x= 233 y= 991 N= 1991
GCD(x,y) using Euclidean algorithm = 1
GCD(x,y) using Extended Euclidean algorithm= 1
Bézout coefficients: -336.0 79.0
Multiplicative Inverse(x,N)( 0 - indicates Inverse does not exist) = 94
x^y mod N = 233
(base) PS D:\Research 1.0\1 Applied Cryptography\Assignment 3>
```

Details of Question 3

Jacobi Symbol(a/N) **jacFunc(a,N)**

Link to code- <https://github.com/zeya2u9/Applied-Cryptography/blob/main/Assignment-3/que3.py>

It asks for a and N from user.

Method:

The Jacobi symbol, (m/n) , is defined whenever **n is an odd number**. It has the following properties that enable it to be easily computed.

- $(a/n) = (b/n)$ if $a = b \bmod n$.
- $(1/n) = 1$ and $(0/n) = 0$.
- $(2m/n) = (m/n)$ if $n = \pm 1 \bmod 8$. Otherwise $(2m/n) = -(m/n)$.
- **(Quadratic reciprocity)** If m and n are both odd, then $(m/n) = (n/m)$ unless both m and n are congruent to 3 mod 4, in which case $(m/n) = -(n/m)$.

If n is a prime, then $(m/n) = 1$ exactly when m is a nonzero square mod n (a quadratic residue).

Output:

```
(base) PS D:\Research 1.0\1 Applied Cryptography\Assignment 3> python que3.py
Constraint:: only upto 64-bit numbers are allowed
Enter a of (a/N):
10
Enter N of (a/N):
91
Jacobi symbol of ( 10 / 91 ) [output as -2 is showing N is invalid]:: -1
(base) PS D:\Research 1.0\1 Applied Cryptography\Assignment 3>
```

Details of Question 4

The Solovay-Strassen and the Miller-Rabin algorithm for primality testing

Link to Code- <https://github.com/zeya2u9/Applied-Cryptography/blob/main/Assignment-3/que4.py>

The Solovay-Strassen method: solStr(n)

Taken from the lecture notes

Choose a random integer a such that $1 \leq a \leq n - 1$

$x \leftarrow \left(\frac{a}{n}\right)$

if ($x = 0$)

return " n is composite"

$y \leftarrow a^{(n-1)/2} \bmod n$

if ($x \equiv y \bmod n$)

return " n is prime"

else

return " n is composite"

The Miller-Rabin algorithm: millRab(n)

Taken from the lecture notes

Write $n - 1 = 2^k m$, where m is odd

Choose a random integer a such that $1 \leq a \leq n - 1$

$b \leftarrow a^m \bmod n$

if ($b \equiv 1 \bmod n$)

return " n is prime"

for $i \leftarrow 0$ **to** $k - 1$

If ($b \equiv -1 \bmod n$)

return " n is prime"

else

$b \leftarrow b^2 \bmod n$

return " n is composite"

These algorithms use modPlus(), modMult(), squAndmult() and jacFunc() methods internally.

Output:

```
Anaconda Powershell Prompt
(base) PS D:\Research 1.0\1 Applied Cryptography\Assignment 3> python que4.py
Constraint:: only upto 64-bit numbers are allowed
Solovay strassen Method
Choose n
991
991 is Prime
Miller rabin method
Choose n
889
889 is Composite
(base) PS D:\Research 1.0\1 Applied Cryptography\Assignment 3>
```

Details of Question 5

Plain RSA Encryption Scheme

Link to code- <https://github.com/zeya2u9/Applied-Cryptography/blob/main/Assignment-3/que5.py>

Algorithm:

1. Choose two prime numbers p and q . For this it first chooses two numbers randomly and checks for their primality using Miller Rabin method from previous question.
2. Calculate $n=p*q$
3. Calculate $\phi(n)=(p-1)(q-1)$
4. Choose e such that $\gcd(e, \phi(n))=1$ i.e.; e and $\phi(n)$ are coprime. For this $\text{GCD}()$ method of que-2 has been used.
5. Calculate d such that, $ed \equiv 1 \pmod{\phi(n)}$. For this $\text{multInverse}()$ method of que-2 has been used.

Output – Call **rsaGen()**

public key $= (n, e)$

private key $= (p, q, d)$

Encryption : Call **rsaEnc(m,e,n)**

$c = \text{Enc}(m)$

$\text{Enc}(m) = m^e \pmod n$, for this $\text{squAndmult}()$ method of que-2 has been used.

Decryption: Call **rsaDec(c,d,n)**

$m = \text{Dec}(c)$

$\text{Dec}(c) = c^d \pmod n$, for this $\text{squAndmult}()$ method of que-2 has been used.

Output::

```
(base) PS D:\Research 1.0\1 Applied Cryptography\Assignment 3> python que5.py
Constraint:: only upto 64-bit numbers are allowed
RSA GENERATION PROCESS:: public private key pairs

Choosing p randomly
First prime p: 14969
Choosing q randomly
Second prime q: 32569
phi_n: 487477824
Value of e: 2243762957
Public key pair(n,e): 487525361 2243762957
Private key pair(p,q,d): 14969 32569 172675013
Choose n - the max limit for taking messages 0<= m <= n-1
888
How many messages you want to encrypt and decrypt?
3
M 0
Choose message m:[an integer between 0 and 887 ]
65
Encrpyted message: 349346231
Now decrypting it
Decrpyted message: 65
M 1
Choose message m:[an integer between 0 and 887 ]
30
Encrpyted message: 466143977
Now decrypting it
Decrpyted message: 30
M 2
Choose message m:[an integer between 0 and 887 ]
780
Encrpyted message: 37187812
Now decrypting it
Decrpyted message: 780
(base) PS D:\Research 1.0\1 Applied Cryptography\Assignment 3>
```