RSA Public-Key Cryptosystem

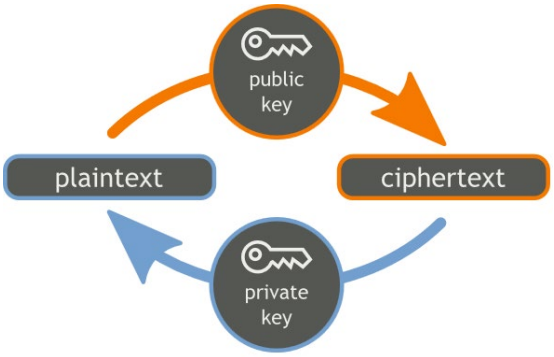
Problem Definition

The goal is to implement the RSA public-key cryptosystem. The RSA (Rivest-Shamir-Adleman) cryptosystem is widely used for secure communication in browsers, bank ATM machines, credit card machines, mobile phones, smart cards, and the Windows operating system. It works by manipulating integers. To prevent listeners, the RSA cryptosystem must manipulate huge integers (hundreds of digits). The built-in C type **`int`** is only capable of dealing with 16 or 32 bit integers, providing little or no security. You will:

1. Design, implement, and analyze an extended precision arithmetic data type (big integer) that is capable of manipulating much larger integers.
2. Use this data type to write a client program that encrypts and decrypts messages using RSA.

Cryptosystem mean encoding and decoding confidential messages.

RSA is a public key cryptosystem algorithm. Public key cryptosystem requires two separate keys, one of which is secret (or private) and one of which is public. Although different, the two parts of this key pair are mathematically linked. The public key is used to encrypt plaintext; whereas the private key is used to decrypt cipher text. Only the one having the private key can decrypt the cipher text and get the original message.



How RSA Works?

Alice wants to send message M to Bob using RSA. How will this happen?

RSA involves three integer parameters **d, e,** and **n** that satisfy certain mathematical properties. The private key (**d, n**) is known only by Bob, while the public key (**e, n**) is published on the Internet. If Alice wants to send Bob a message (e.g., her credit card number) she encodes her integer **M** that is between 0 and **n**-1. Then she computes:

$$E(M) = M^e \bmod n$$

and sends the integer **E(M)** to Bob. When Bob receives the encrypted communication **E(M)**, he decrypts it by computing:

$$\mathbf{M = E(M)^d \bmod n.}$$

Demo: Check the demo here

Example:
e = 7, d = 2563, n = 3713 and the message M is 2003

Alice computes:

$E(M) = 2003^7 \bmod 3713 = 129{,}350{,}063{,}142{,}700{,}422{,}208{,}187 \bmod 3713 = 746.$

Bon receives 746 so he can compute the original message as follows:

$M = 746^{2563} \bmod 3713 = 2003.$

Goal and Suggestions

Your goal is to implement the BigInteger data type and use it in RSA public-key cryptosystem so you can be able to encrypt and decrypt large integer **M**.

The RSA cryptosystem is easily broken if the private key **d** or the modulus **n** are too small (e.g., 32 bit integers). The built-in C types **`int`** and **`long`** can typically handle only 16, 32 or 64 bit integers. Your main challenge is to design, implement, and analyze a BIG INTEGER that can manipulate large (nonnegative) integers. Your data type will support the following operations: addition, subtraction, multiplication, division, and mod of power.

1- The first and most critical step is to choose an appropriate data structure to represent N-digit big integers. A natural approach is to store each digit as an element in an array.

2- You need to implement this BigInteger class along with its basic functions: addition, subtraction, multiplication, division, and mod of power. Remember that you're not working with **`int`** or **`long`**

   Efficient Multiplication. Implement the efficient integer multiplication (less than N^2) using the **Karatsuba algorithm**. It should take two big integers as input and return a third big integer that is the product of the two. Observe that if **a** and **b** are N-digit integers, the product will have at most 2N digits.

   Division. Integer division is the most complicated of the arithmetic functions. Here is one of the simplest algorithms to compute the quotient **q = a / b** and the remainder **r = a mod b**, where **a** and **b** are big integers, with **b** not equal to 0.

```
div(a, b)
{
    if (a < b)
        return (0, a)
    (q, r) = div(a, 2b)
    q = 2q
    if (r < b)
        return (q, r)
    else
        return (q + 1, r - b)
}
```

Mod of Power . It is used to calculate:

$$R = B^P mod\ M$$

Direct calculation of $B^P\ mod\ M$ take O(P) for integer values. However, we can make use of the following theory to reduce its complexity.

(A × B × C) mod N = **[**(A mod N) × (B mod N) × (c mod N)**]** mod N

Since we want **B** to the power **P** and take modulus **M**, so it is going to be:

(**(B** mod **M)** × **(B** mod **M)** × **(B** mod **M)** × ...*(**P** times)*) mod **M**

Hint: You can think of it using D&C

3. After that you'll design and implement an **<u>efficient</u>** algorithm for the RSA encryption and decryption functions as explained before.

Project Requirements

Required Implementation

Requirement	Performance
1. Big integer class	-
2. Add function for 2 big integers.	**Time:** should be **bounded by O(N)**, N is the number of digits of the big int
3. Subtract function for 2 big integers.	**Time:** should be **bounded by O(N)**, N is the number of digits of the big int
4. Multiply function for 2 big integers.	**Time: LESS THAN O(N^2)**, N is the number of digits of the big int
5. Div function for 2 big integers.	**Time:** should be **bounded by O(N)**, N is the number of digits of the big int
6. Mod of power function for 2 big integers.	**Time:** should be **bounded by O(N^2 Log(P))**, N is the number of digits of the divisor and P is the number of digits of the power
7. Encrypt function of the RSA.	**Time:** should be **bounded by O(N^2 Log(P))**, N is the number of digits of the divisor and P is the number of digits of the power
8. Decrypt function of the RSA.	**Time:** should be **bounded by O(N^2 Log(P))**, N is the number of digits of the divisor and P is the number of digits of the power

Input

File containing **number of test cases** in the first line followed by each test case that is represented in 4 lines as follows:

1. N
2. e or d
3. M or E(M)
4. 0 or 1 (0-> encrypt, 1-> decrypt)

Output

1. Save the result of each encryption or decryption process in a single file each result on a separate line.
2. Calculate the execution time encryption/decryption

How to calculate execution time?

To calculate time of certain piece of code:

1. Get the system time before the code
2. Get the system time after the code
3. Subtract both of them to get the time of your code

To get system time in milliseconds using C#, you can use `System.Environment.TickCount`

Test Cases

- Sample Case:
  - 10 test cases with values you can trace. Not very large integers. To be able to check your output, consider each 2 test cases together, the first is the encryption and the second is its decryption.

- Complete Test:
  - 6 test cases with big integers.

Deliverables

Implementation (60%)

1. `BigInteger` Class with its functions
2. RSA Encryption
3. RSA Decryption

Document (40%)

1. Source code of the `BigInteger` with add, sub, div, mul and mod of power functions
2. **DETAILED ANALYSIS** of add, sub, div, mul and mod of power functions
3. Execution time of "COMPLETE Test" Cases only

Allowed Codes [if any]

- No other external code is allowed.

Milestones

	Deliverables	Due to
Milestone1	**1.** BigInteger class with add, sub & mul functions **2.** Documentation I	At the END of WEEK 12 [Schedule will be announced]
Milestone2	**1.** Div and mod of power **2.** Encrypt & decrypt functions **3.** Documentation II	Final Delivery Lab Exam week
For Milestone1: ▪ **MUST** deliver the required tasks and **ENSURE** it's worked correctly ▪ **MUST** deliver the **part of the documentation** that is related to the Milestone (printed document)		

BONUSES

Level 1:

Adjust RSA to work with strings rather than integers. Note that string size in unknown.

Level 2:

Implement a function that generates the public key (n,e). You need to:

1. Compute two large pseudo-random **prime** numbers ***p*** and ***q*** of a specified number of digits.
2. Compute $\varphi = (p - 1)(q - 1)$, and
3. Select a small integer ***e*** that is relatively prime with φ. Use these to generate a public key (e, n)