Intermediary report of RSA implementation using C++

Prepared by Md Rakib Shahriar (010754531)

Steps of RSA

The Implementation of RSA Cryptography algorithm involves following basic steps:

- Generate two different primes p and q
- Calculate the modulus $N = p \times q$
- Calculate the totient $\varphi(n) = (p - 1) \times (q - 1)$
- Select public exponent e such that $1 < e < \varphi(n)$ and $\gcd(\varphi(n), e) = 1$
- Calculate private exponent d such that $d = e^{-1} \bmod \varphi(n)$
- Public Key = [e, n]
- Private Key = [d, n]

Implementation Details

String data object from C++ string library is used as the data structure. Each digits of BigIntegers are characters that are 8 bits in length.

Algorithms

Algorithm of Addition:

Input (BigInteger n, BigInteger m)

Balance the length of the two big integers by putting zeros from left

Loop:

Start from LSB to add digits

If there is a carry

Add to the next digit of the first number

End Loop

If added MSB is greater than 10 then insert a new character as the MSB in the sum

Return sum

Algorithm of Subtraction:

Input (BigInteger n, BigInteger m)

Balance the length of the two big integers by inserting zeros from left

Loop:

Start from LSB to subtract digits

If there is a carry

Add to the next digit of the second number

End Loop

Remove zeros if there is any on the left of the subtracted big integer

Return result

Algorithm of Multiplication:

Input (BigInteger n, BigInteger m)

If sizeof (n) < sizeof(m) then swap(n, m) to reduce calculation

Initiate BigInteger result

Loop on length of m:

Start with LSB of m and take digit d

Loop on length of n:

Multiply n with d

End Loop

Add multiplication to result

End Loop

Return result

Algorithm of Division:

Input (BigInteger dividend, BigInteger divisor) *if dividend is less than divisor*

then we don't need to divide at all; return (0, dividend)

Loop until remainder is greater than or equals to divisor

check if divisor and remainder are equal sized

If yes then quotient should be less than 10

If not equal sized then project a close multiple of the divisor to dividend by multiplying 10's

Take first sizeof (divisor) number of digits from dividend and form a BigInteger

Check if the formed BigInteger is greater than divisor

If yes then multiply divisor to project the closest number to formed BigInteger and calculate number of zeros required to append at the end of divisors multiple

If no then multiply divisor to project the closest number to formed BigInteger and calculate number of zeros (less one than the If yes block) required to append at the end of divisors multiple

Append ratio to the partial quotient

Append zero's to the partial quotient

Add partial quotient to the quotient

Subtract the multiplied number to the remainder

End of loop

Return (quotient, remainder)

Algorithm of Modular:

Check if modulo by 2

If yes then check the last digit of the dividend and return 0 or 1

Check if module by 3

If yes then sum all the digits of dividend; return sum%3

Else then call division function and calculate the remainder

Return remainder

How to Run the Program

The program currently takes fixed 1000 bit as length of N. As the function of random generation of Big prime integer is yet to be done, two fixed 151 digits long prime numbers are taken as p and q. After the program is run, please input a numerical number. Sometimes it is throwing exception for string inputs. A workable input would be: 1387217321**.**

Time comparison with Java for basic arithmetic operations

A Java program is developed using the BigInteger class. The following comparison is found running the developed C++ BigInteger and Java BigInteger in a single machine.

Operations	**C++**	**Java**
+	0.02 ms	0 ms
-	0.038 ms	0 ms
*	4.183 ms	0 ms
/	0.077 ms	0 ms
%	0.088 ms	0 ms

<u>**Time comparison with java for algorithms to calculate RSA:**</u>

# of Bit of N	**RSA Steps**	**C++**	**Java**
1000	Calculate N	0.544 ms	0ms
	Calculate e	14.812 ms	0ms

	Calculate d	19.248 ms	9 ms
	Encryption	0.00027 s	0 ms
	Decryption	25.5768 s	7 ms
2000	Calculate N	1.818 ms	0ms
	Calculate e	20.719 ms	0ms
	Calculate d	48.713 ms	11 ms
	Encryption	0.000193 s	1 ms
	Decryption	148.406 s	49 ms
3072	Calculate N	8.509 ms	0 ms
	Calculate e	45.043 ms	0 ms
	Calculate d	90.48 ms	9 ms
	Encryption	0.000488 s	1 ms
	Decryption	701.199 s	72 ms

Future Work

- Improvement of multiplication and division algorithm
- Random prime number generation
- Test and evaluate performance for larger size of N

References

[1] https://github.com/panks/BigInteger