Supporting Documentation for RSA cryptosystem program

Written by- Utsav Poddar

Date- 2/18/2017

What is RSA cryptosystem?

RSA cryptosystem is an asymmetric cryptographic algorithm which derives its unbreakable power from the difficulty of prime factorization of really large numbers.

Language used: Java

Data constraints handled:

1. Message Length: 4095 bits ~= 12 bytes
2. Key Length: 4096 bits (current industry standard)

Procedure:

1. Choose two random primes, p and q, greater than 100 digits long (The program has 617 digits support)
2. Compute n=p*q. n is used as modulus for both public and private key. Its length is the key length.
3. Compute phi(n), i.e. the Euler Totient Function for n which is a co-prime of n. phi = (p-1)*(q-1)
4. Choose public key "e" such that 1<e<phi(n) and GCD(e,phi(n)) = 1, i.e. they are co-primes. Short bit length and small hamming weight results in more efficiency, most commonly 2^16 - 1 = 65537.
5. Determine d as d = inverse(e) mod(phi(n)), i.e. modular multiplicative inverse of e(mod phi(n))

Encryption:

The RSA algorithm employs the use of modular arithmetic for encryption. The formula for encrypting the message is: c(encrypted message)=(m^e) modulo n .

Decryption:

The RSA algorithm employs the use of modular arithmetic for decryption. The formula for encrypting the message is: m(original message)=(c^e) modulo n .

These fact is supported by the Euler totient function and inverse modular arithmetic rule, Euler's theorem which states that any number raised to the power phi(n) modulo n will be equal to one. Mathematically it says:-

m^phi(n) = 1 mod(n)

Handling constraints:

The BigInteger class of Java has been used to write the code for RSA encryption mainly because it can handle really big numbers without any trouble. To keep in mind the security of underlying algorithm, a lot of methods have been employed such as SecureRandom and ProbablePrime which ensure that prime generation is completely random(the java.math.random class employs algorithms that generate pseudorandom numbers which are not fit for cryptography purposes).

Design Considerations:

The file has been kept in two parts, the first one(namely RSA.java) contains all the code for random keys generation and further on, i.e. from step 1 to step 5, the reason for doing so is to enable easy distribution amongst clients later who would need only the implementation, not details of working. This way abstraction and encapsulation have been used.

The variables in RSA have been made private so that none of the other classes can use them, this is very important for security of the cryptosystem.

The method encrypt has been made public with the belief that any code that runs on a client’s machine would be in a different package and hence would be able to access the public method encrypt.

The method decrypt, for the same reason has been kept default as it would be used by server’s local machine and it has been assumed that server code runs in this same directory. So only the server would be able to call the decrypt method and hence adds to extra security.

The driver file, namely RSA_driver.java has all along used methods described by the RSA.java file and this is therefore ensuring extra security, injection attacks would be prevented.

1. The program first generates a random number which is 4095 bits long (user input would also work here).
2. It calls the encrypt method.
3. Displays the encrypted message.
4. Calls the decrypt method.
5. Displays the original method.
6. Shows the time required for both encryption and decryption.

**Kindly note that this is only a test/driver program, actual program would have increased overheads and complexities as there would be communication time, importing time and a few other factors.