Table of Contents

First of all

This is my operating system curriculum design, I wrote it just because I have to chose one topic to wrote. So I chose the topic is an analog implementation of a public encryption algorithm. I do know that there is no difficulty, but there is no difficulty in other topics. It's here for the convenience of those who want to know how RSA(encryption algorithm) works.

Operation

See the wiki for details.

In simple terms, there are five steps that require generating three numbers.

1. Generate two unequal large prime, denoted as p and q.
2. Calculate $n = pq$.
3. According to the Euler function, calculate $\varphi(n)$, denoted as r.

$$r = \varphi(n) = \varphi(pq) = \varphi(p)\varphi(q) = (p-1)(q-1)$$

4. Generate a number e and e is coprime to r, and $1 < e < r$.
5. Generate a number d and $ed \equiv 1(\mod n)$, which means $(ed-1) \mod n = 0$, d is the modular multiplicative inverse of e about n.

Then (n, e) is public key, and (n, d) is private key.

Let m be the information that needs to be encrypted, and record c an the encrypted cipertext(m and c are both large numbers smaller than n).

then

$$c = m^e \mod n$$
$$m = c^d \mod n$$

Programming Implementation

Implemente the aboves steps in java via the BigInteger class.

1. For generating large prime, Use `new BigInteger(BitLength, Certainty, Random)`
2. For generating d, Use `e.modInverse(n)`.
3. For calculae c and m, Use `m.modPow(e, n)` and `c.modPow(c, n)`.

For those who want to implement all by themselves.

See those links.

1. For testing if a integer is a prime, see Fermat's little theorem
2. For calculating modular multiplicative inverse of e about r, see extended Euclidean algorithm
3. For calculating modulo operation of high-order of large integer, see Montgomery modular multiplication

Other algorithms may also work.