Number theory facts needed for the RSA algorithm

1. If \(a\) and \(b\) are positive integers, then \(\text{gcd}(a, b)\) is the greatest common divisor of \(a\) and \(b\). For example, \(\text{gcd}(30, 42) = 6\). This is because the divisors of 30 are: 1, 2, 3, 5, 6, 10, 15, and 30, and the divisors of 42 are: 1, 2, 3, 6, 7, 14, 21, and 42. Thus, the greatest common divisor is 6. The function \(\text{gcd}(a, b)\) is also sometimes denoted by \((a, b)\). There is a class \(\text{BigInteger}\) in Java that handles all sort of operations on “essentially” arbitrarily large integers. It easily handles numbers with several hundred digits. This class has an instance method \(\text{BigInteger gcd(BigInteger n)}\); which computes the greatest common divisor of \(a\) and \(n\).

   There is an extremely efficient algorithm to calculate \(\text{gcd}\) of two numbers. It is based on the following observation: Let \(a > b > 0\) be two integers, and write
   \[
   a = bq + r, \quad 0 \leq r < b
   \]
   i.e., we divide \(a\) by \(b\) and calculate the quotient and the remainder. Then it is easy to see that the list of common divisors of \(a\) and \(b\) is the same as the list of common divisors of \(b\) and \(r\). Another words, \(\text{gcd}(a, b) = \text{gcd}(b, r)\). We then repeat the procedure, until we have to calculate \(\text{gcd}(k, 0)\) for some \(k > 0\), and the answer to that is just \(k\). This idea goes back to Euclid. It is one of the most successful and useful algorithms of all times.

2. If \(a\) and \(m\) are two positive integers, then \(a \pmod{m}\) is the remainder resulting from dividing \(a\) by \(m\). Thus, \(13 \pmod{5} = 3, 21 \pmod{4} = 1\), etc. The operation \(\pmod{\phantom{0}}\) is denoted in C/C++ by \(\%\). Thus, \(24 \% 7\) results in the value 3. Some complications arise when some or all of the arguments are negative, we will ignore the issue. There are two methods in the class \(\text{BigInteger}\) which handle these matters: \(\text{modulo}\) and \(\text{remainder}\). In both cases they calculate \(a \pmod{m}\), the differences occur when one or both arguments are negative. Useful properties of \(\text{mod}\) function are:
   \[
   \begin{align*}
   (a + b) \pmod{m} &= (a \pmod{m} + b \pmod{m}) \pmod{m} \\
   a \times b \pmod{m} &= (a \pmod{m}) \times (b \pmod{m}) \pmod{m}
   \end{align*}
   \]
   For example:
   \[
   (137 + 578) \pmod{53} = (137 \pmod{53} + 578 \pmod{53}) \pmod{53} = 26
   \]
   That is: \(715 \pmod{53} = (31 + 48) \pmod{53} = 79 \pmod{53} = 26\). Similarly:
   \[
   (137 \times 578) \pmod{53} = (137 \pmod{53} \times 578 \pmod{53}) \pmod{53}
   \]
   That is \(79186 \pmod{53} = 31 \times 48 \pmod{53} = 1488 \pmod{53} = 4\). (Check it all out by yourself) This means that we can do arithmetic \(\pmod{m}\), just like we do ordinary arithmetic. (There are some problems with division, when \(m\) is not a prime number, but we will not go into this.)
3. Two numbers $a$ and $b$ are called relatively prime if $\gcd(a, b) = 1$. If $\gcd(a, b) = 1$, there are two positive integers $x$ and $y$ such that:

$$ax = 1 + by$$

The proof that this is so, and an efficient method of finding $x$ and $y$ can be done using the Euclid algorithm. In particular, if $\gcd(a, m) = 1$, then there is an integer $x$ such that:

$$ax = 1 \pmod{m}.$$ 

Hence if $m$ is a prime number, every number $a \not\equiv 0 \pmod{m}$ has an inverse in the multiplication $\pmod{p}$. If $m$ is not a prime number than only those $a$'s which are relatively prime to $m$ have the multiplicative inverse. There is a method $\text{BigInteger modInverse(BigInteger n)}$; which returns the inverse of this $n$ modulo, if it exists. It throws an exception if there is no

4. If $n > 0$ is a positive integer, we denote by $\phi(n)$ the number if integers $k$ such that $k < n$ and $\gcd(k, n) = 1$. For example, $\phi(12) = 4$. This is because if you take all the numbers from 1 to 11, only four of them: 1, 5, 7, and 11 are relatively prime to 12. As another example, $\phi(15) = 8$: The numbers less than 15 and relatively prime to 15 are: 1, 2, 4, 7, 8, 11, 13, and 14 – there are 8 of them. In general, given a large $n$, it is difficult to find $\phi(n)$. However, if we know that $n = p \cdot q$, and $p$ and $q$ are prime numbers, then $\phi(n) = (p - 1) \cdot (q - 1)$. The theorem that makes public key cryptography possible is as follows:

**Theorem (Euler).** Suppose $\gcd(a, n) = 1$. Then $a^{\phi(n)} \equiv 1 \pmod{n}$.

A very rough idea how this is used in cryptography is as follows: Suppose we want to encrypt a number $x$. To this end, two large primes $p$ an $q$ are chosen, and the number $N = p \cdot q$ is calculated and made public ($p$ and $q$ are kept secret). We arrange things so that $N > x$. A number $E$, relatively prime to $\phi(N)$ is also calculated, and also made public. To encode the number $x$, we calculate $y = x^E \pmod{n}$. To recover the message $x$, find a number $D$ such that $E \cdot D = 1 \pmod{\phi(N)}$, i.e., $E \cdot D = 1 + \phi(n) \cdot x$ for some integer $x$. This is relatively easy, using the Euclidean algorithm. Then

$$S^D = M^{E \cdot D} = M^{1 + \phi(n) - x} = M \cdot M^{\phi(n) - x} = M \pmod{n} = M$$

Thus, we reconstructed $x$. Notice that a bad guy could also do the same thing, provided he can find $p$ and $q$. If, however, $n$ is large (200+ digits) there is no way, at the present time, to do it in any reasonable amount of time. (Before the solar system expires, say).

There is a question of how to compute $a^b \pmod{n}$ for large values of the arguments. There is a method $\text{BigInteger modPower(BigInteger b, BigInteger n)}$; which calculates this $b \pmod{n}$, but the implementation is far from obvious. The method works when all the arguments are hundreds, if not thousands digit long, so straightforward idea of computing $a^b$ by multiplying $a \cdot a \cdot \ldots \cdot a$ ($b$ times) and then computing $\pmod{n}$ is completely impractical. The basic idea is as follows. Notice that
This leads to a recursive definition of \( f(a, b, m) = a^b \mod m \), in pseudocode:

\[
 a^b = \begin{cases} 
 (a^2)^{\frac{b}{2}} & \text{if } b \text{ is even} \\
 a \cdot a^{b-1} & \text{if } b \text{ is odd}
\end{cases}
\]

It actually works pretty good and can be implemented by most SJSU students, at least those taking Scheme from me.

5. Security. Now consider an unauthorized person trying to intercept the message. The public quantities are \( y \) (The encrypted message), the modulus \( N \), and the encryption exponent \( E \). To recover the original message \( x \), called plaintext, all we have to do is to figure out the decryption algorithm \( D \). That can be easily computed, using the modInverse method provided by Java as well as numerous other sources, provided that we know the numbers \( p \) and \( q \). Thus, the security lies in the fact that we cannot factor large integers.