Notes on the RSA Algorithm

The RSA algorithm is based upon the difficulty of finding the prime factorization of numbers whose prime factors are large primes—say 100 digit prime numbers. The RSA algorithm works as follows:

1. Select two prime numbers \( p \) and \( q \).
2. Form \( n = pq \). Note that \( n \) is public and can be published.
3. Form \( y = (p - 1)(q - 1) \). Note that \( y \) is secret and not published.
4. Select an integer \( e \) such that \( e < n \) and the \( \gcd(e,y) = 1 \). Note the author’s requirement that \( e \) not divide \( y \) evenly is not strong enough. For the algorithm to work, \( e \) and \( y \) must be relatively prime-\( \gcd(e,y) = 1 \).
5. To encode a letter use its ASCII value. For instance ‘A’ is at position 65 in the ASCII sequence, so it is encoded as:

\[
c = 65^e \mod n
\]

The letter to be encoded is said to be a plaintext letter and the message before it is encoded is said to be a plaintext message.

6. To decode a character, you must first calculate a value \( d \) that satisfies the equation:

\[
e \cdot d \mod y = 1 \quad \text{(Eq. 6.1)}
\]

where \( y \) and \( e \) are the values calculated in 3 and 4 above, respectively. Since \( e \) and \( y \) are relatively prime, we know that there exist integers \( d \) and \( g \) such that

\[
e \cdot d + y \cdot g = 1 \quad \text{(Eq. 6.2)}
\]

We can solve this equation by using the Euclidean Algorithm with back substitution. To illustrate the process, suppose we choose \( p = 11 \) and \( q = 13 \). Then \( n=143 \) and \( y = 10 \cdot 12 = 120 \). Further, suppose that we select \( e=19 \). Note that \( e \) and \( y \) are relatively prime, \( \gcd(19,120)=1 \).

To find \( d \) proceed as follows using the Euclidean Algorithm as if we were looking for the solution to the \( \gcd(19, 120) \).

\[
\begin{align*}
120 &= 6 \cdot 19 + 6 \\
6 &= 120 - 6 \cdot 19 \\
19 &= 3 \cdot 6 + 1 \\
1 &= 19 - 3 \cdot 6
\end{align*}
\]

The values of \( d \) and \( g \) are found by working backwards from the last equation in the right column. The process is to substitute using the value from the equation the right column immediately above, simplify, and repeat until you get back to equation Eq. 6.2.

\[
1 = 19 - 3 \cdot (6) \\
= 19 - 3 \cdot (120 - 6 \cdot 19)
\]
\[
= 19 - 3 \cdot 120 + 18 \cdot (19) \\
= 19 \cdot (19) - 3 \cdot (120)
\]
Note that this equation is in the form \( e \cdot d + y \cdot g = 1 \), where \( d = 19 \) and \( g = -3 \).

The range for the mod \( y \) function (Modulus \( y \) arithmetic) is defined to be the interval \( 0 \ldots y \). So, had \( d \) come out to be negative we would convert it to a positive integer by adding \( y \), that is we would use the value \(-d+y\) for \( d \). In this example there is no need to do so since \( d \) is already positive. Also note that \( d \) just happened to equal \( e \) in this example. In general this will not be true.

7. Using the value of \( d \), as calculated in step 6, the plaintext character is decoded as:

\[
m = c^d \mod n
\]

where \( c \) is an character that has been encoded as described in 5 above.

**Example:** From 6 above we have \( p = 11, q = 13, n = 143, y = 120, e = 19 \) and \( d = 19 \). To encode the ASCII letter H (value 72) we calculate the encrypted character, \( c \), as:

\[
c = 72^{19} \mod 143 = 123
\]

Decoding \( c \) using \( d \) we have

\[
m = 123^{19} \mod 143 = 72.
\]

**Some Java Notes:** The values generated using the RSA algorithm for even small primes like 11 and 13, are larger than can be represented as a Java long. For instance \( 72^{19} \) is much larger than the largest long integer, \( 2^{64} - 1 \). Fortunately, the Java libraries provide a solution to this dilemma. The java.math package contains a class named BigInteger that can be used to represent an integer of arbitrary precision. The following code illustrates using the BigInteger class. Note the use of the Integer wrapper class also.

```java
int e = 19;
int n = 143;
int d = 19;
BigInteger plainText;
BigInteger bigN = new BigInteger(new Integer(n).toString());
BigInteger cypherText;

String curCharVal = new Integer((int)'H').toString();
System.out.println("Character H is: " + curCharVal);

plainText = new BigInteger(curCharVal);
// encode the character
plainText = plainText.pow(e);
cypherText = plainText.mod(bigN);
```