CMSC 222 Project 2: RSA

Due Date: 11:59pm, Fri. Oct 19

Overview:
In this project, you will write code to implement the RSA public key cryptographic algorithm, and you will perform experiments using this algorithm.

Project Description:
The RSA (Rivest, Shamir, Adleman) public key cryptographic algorithm performs encryption as well as decryption. RSA works by use of two keys --- a public key and a private key. Using your public key, anyone can encrypt a message and send it to you. In the event that an adversary intercepts the encrypted message, she cannot decipher the message unless she knows your private key (reverse engineering with the public key will not work here). Encryption and decryption under RSA are based on modular exponentiation using large prime numbers, as described next.

To generate a public key, choose two large prime numbers $p$ and $q$ (in practice, these are typically around 256 bits each) and let $n = pq$. Then choose a number $e$ that is relatively prime to $(p-1)(q-1)$. The public key is $<e,n>$. To encrypt her plaintext message (or at least a portion thereof) represented as an integer $M$ (you must have $M < n$), your friend Alice can compute the ciphertext $C = M^e \mod n$. Only you will be able to decrypt $C$ using your private key.

To generate a private key, find the unique number $d$ that is the multiplicative inverse of $e$ modulo $(p-1)(q-1)$ – this process involves the Euclidean algorithm and was discussed in class. The private key is $<d,n>$. You can then compute the original message (or portion thereof) $M = C^d \mod n$ using Alice’s encrypted message $C$.

Unless you do something silly (like give away your private key), RSA is by current standards a very secure means of encrypting information. More specifically, we have no guarantee that RSA is in fact secure. We can only trust that many smart people have been attempting to break RSA, but they have been unsuccessful because factoring a large number is hard
1 (Remember that we choose $n = pq$ where $p$ and $q$ are large primes, so unless the adversary can factor $n$, she cannot determine the exponent $d$ that will allow her to decrypt messages.)

So how do we choose large primes $p$ and $q$? We could try the Sieve of Eratosthenes, but for very large primes this will be extremely inefficient (in fact, prohibitively so). The method usually used is beyond the scope of this project, but it will suffice to say that we choose a large number and then test its primality probabilistically. If the number chosen passes the test, with very high probability the number will actually be prime and therefore

1. We do not know that factoring $n$ is the only way to break RSA, but we do know that breaking RSA is no more difficult than factoring.
RSA will work. It should be noted that in practice public key cryptography is not typically used to encode and decode entire messages. Instead, public key cryptography is often used to send a private key so that parties can then communicate using private key cryptography, which is in general less computationally intensive. Nonetheless, we will use RSA public key cryptography to encode and decode messages for this project – a great application of using the Euclidean algorithm and prime number and modular arithmetic concepts presented in class.

Specifications:
1. You must implement your code in Java.
2. You must produce three separate executables: rsaEncrypt, rsaDecrypt, and gcd.
3. rsaEncrypt must accept as command line arguments:
   3.1. an input string, delimited by double quotation marks, of ASCII characters to be encoded;
   3.2. a public key exponent \(e\);
   3.3. and a large number \(n\) (for this project, but not in practice, it will suffice to store \(n\) as a long).
4. rsaDecrypt must accept as command line arguments:
   4.1. an input string of digits (the output from rsaEncrypt, for example);
   4.2. a private key exponent \(d\);
   4.3. and a large number \(n\).
5. gcd must accept as command line arguments two integers on which to perform the Euclidean algorithm as discussed in class.

Encrypting:
Your program rsaEncrypt must perform the RSA encryption algorithm on the provided input string of ASCII characters using the given public key \(<e,n>\) where \(n = pq\). More specifically:
1. Represent each character in the input string by its 3-digit ASCII numeric equivalent (e.g., ‘!’ is 033, ‘C’ is 067, ‘z’ is 122) – cast the char to an int.
2. Break the resulting sequence of numbers into blocks of 8 – each block will comprise the \(M\) in the \(C = M^e \mod n\) operation. (We will use blocks of 8 for this project because the values of \(n\) that we will use will be 9 digits long. Remember from above that we must have \(M < n\).) If the final block is fewer than 8 digits, pad to the right with sufficient zeros. Note that it is easier to pick out digits if your 8 digit blocks are actually stored as Strings of digits rather than longs. You can convert back to numeric type using Long.parseLong before doing the modular exponentiation.
3. For each block of 8 digits, perform the encryption algorithm on that block. The result will be a number less than \( n \). If the number is less than 9 digits (the length of \( n \) for this project), pad to the left with sufficient zeros.

4. Concatenate all the 9-digit encrypted results and output the entire string.

Your program must produce output similar to the following:

```
% java rsaEncrypt "CMSC 222" 13 100160063
032059734076216682096586161
```

The above example uses \( e = 13 \) and \( n = 100,160,063 \) (i.e., \( p = 10,007 \) and \( q = 10,009 \)). Note that \( \gcd(13, 100,140,048) = 1 \) where \( (p-1)(q-1) = 100,140,048 \), i.e., \( e \) and \( (p-1)(q-1) \) are relatively prime.

Decryption:

Your program \texttt{rsaDecrypt} must perform the RSA decryption algorithm on the provided input string of digits using the given \textit{private} key \( <d,n> \) where \( n = pq \). More specifically, reverse the process from above:

1. Break the input string of digits into blocks of 9, and perform the decryption algorithm on each block using \( M = C^d \mod n \).
2. If the result \( M \) is less than 8 digits, pad to the left with sufficient zeros.
3. Concatenate all the 8-digit decrypted numbers.
4. Break this long string of numbers into blocks of three. Convert each 3-digit numeric equivalent to the corresponding ASCII character -- cast the \texttt{int} to a \texttt{char}. Remember that in the encryption process, the final 8-digit block may have been padded with zeros to the right -- do the appropriate thing here.
5. Concatenate all the decrypted characters and output the entire string, delimited by < and >.

Your program must produce output similar to the following:

```
% java rsaDecrypt 032059734076216682096586161
61624645 100160063
<CMSC 222>
```

The above example uses \( d = 61,624,645 \) and \( n = 100,160,063 \) (i.e., \( p = 10,007 \) and \( q = 10,009 \)) because \( d = 61,624,645 \) is the inverse to \( e = 13 \) modulo \( (p-1)(q-1) = 10,140,048 \) as determined by the Euclidean algorithm (see below).

**Euclidean Algorithm:**

Your program \texttt{gcd} must perform the Euclidean algorithm on the two provided input integers. Your program must produce output similar to the following:

```
% java gcd 13 100140048
\texttt{gcd}(13,100140048) = 1 = -38515403(13) + 5(100140048)
```

i.e., in the format \( \text{gcd}(m,n) = r = u(m) + v(n) \) where \( r \) is the final remainder and
the $u$ and $v$ parameters are the same as those discussed in class. The above example shows that, in fact, $e = 13$ is relatively prime to $(p-1)(q-1) = 100,140,048$. As discussed in class, we can write the remainder $1 = r = um + vn$. Here, $u = -38,515,403$ and $v = 5$.

Note that $[-38,515,403 \mod 100,140,048 = 61,624,645 \mod 100,140,048]$ (just add $u$ and $n$) and so we get for $e = 13$ that the inverse is $d = 61,624,645$ (see decryption example above).

**Experiments:**

To make sure that you implement the algorithms correctly, I am requiring you to determine the appropriate decryption exponent $d$ for a given public key pair $<e,n>$, and use $d$ to decode an encrypted message sent to you. Assume that the exponent for encryption is $e = 65,537$. (You typically want $e$ to be smaller than $d$ because, if $d$ is small, an adversary could simply search for small values for $d$ until she finds the right one. It turns out that it doesn’t really matter too much what value is chosen for $e$; 65537 is popular because it is large enough to defeat some easy attacks but because it is $2^{16}+1$, there are only 2 1 bits in its binary representation, so the modular exponentiation algorithm executes relatively quickly.) To facilitate this process, your specific prime $p$ and $q$ values and your encrypted message will be provided in an email at a later date (again, appreciate that in practice these numbers are typically 256 bits each).

I will email your secret message to you individually. In your write-up (see below), make sure to clearly denote your decryption exponent $d$ and the associated decrypted message.

**Write-Up:**

With this project, you must supply a well-written logical discussion of your program with sufficient detail so that – by reading this discussion but not looking at your program – another student in the class could construct a correct program and understand the theory on which it is based. At a minimum, the write-up must include discussion of the algorithm(s) you used in your implementation and any associated underlying mathematics. Also include the results of your experiment outline above.

A sample project and associated write-up are provided on the course web page to give you an idea of what I expect here.

**Submission:**

Email to lbarnett@richmond.edu as attachments your three Java source code files by 11:59pm, Fri., Oct. 19

Consistent with the guidelines provided in the course syllabus,

1. programs that do not compile will receive no credit;
2. programs that do not execute and/or terminate without crashing will receive no credit;
3. any submission not received by the deadline will not be accepted.
Your work on this and all projects in this course is subject to the conditions of the Honor Code as described in the course syllabus.