CS 340 - Software Design

Machine Problem 3, Spring 2013

RSA Encryption/Decryption

Due: Tuesday, March 19, 2013 at 11:59 pm

For this project, you will write a program that will perform a number of operations related to the RSA Encryption/Decryption Algorithm. These operations will include

- creating a public-key/private-key pair,
- "blocking" a large message into manageable sized chunks and its reverse, and
- performing the encrypting/decryption operation.

A Huge-Unsigned-Integer Class

The first thing your program must do is to create a class that can hold very large unsigned integer values. In order to RSA to really be secure it requires the use of prime integer values that have at least 100 decimal digits in them. While we will not be requiring the use of numbers that are nearly that large, we will be requiring numbers that fall outside the range of the primitive C/C++ types.

The idea is to create a coded decimal class in which each decimal digit is stored in its own position in a dynamic array. Thus to store a 50 digit decimal value, we would need an dynamic array of 50 positions. Note that the i^{th} position in the array is storing the decimal digit at the 10^i-th place. Recall the value of:

$$54932 = 5 * 10000 + 4 * 1000 + 9 * 100 + 3 * 10 + 2 * 1$$
$$= 5 * 10^4 + 4 * 10^3 + 9 * 10^2 + 3 * 10^1 + 2 * 10^0$$

So in an array arr[0] = 2
arr[1] = 3
arr[2] = 9
arr[3] = 4
arr[4] = 5

For this class, you are REQUIRED to overload the operators of addition, subtraction, multiplication, division, modulus, assignment, and all of the relational operators. You may overload additional operators as you would like. It is also suggested that you create a constructor that takes an integer value as its parameter. Also input/output methods are suggested.

The RSA Algorithm

Links to pages that discuss this algorithm include the following.

- http://pajhome.org.uk/crypt/rsa/rsa.html (Note, this page gives a very high level discussion of the RSA Algorithm, but has parts that are unclear/confusing when looking at the details.)

- http://www.di-mgt.com.au/rsa_alg.html
- http://mathcircle.berkeley.edu/BMC3/rsa/node4.html

The RSA algorithm needs three related numbers to work. These numbers will be called e, d and n. The pair (e,n) compose the public-key, while the pair (d,n) compose the private key. To encrypt a value M into C we perform the operation of:

$C = M^e \bmod n$

To decrypt the value of C back to M, we perform the operation of:

$M = C^d \bmod n$

The creation of the values of e, d and n starts with the selection of two prime numbers p and q. We need to make sure that the values p and q are large enough to properly encrypt the values we will be using. If we would just be using the ASCII values, the range would be from 0 to 127. Thus to encrypt an ASCII value we need values of p and q such that p * q is larger than 127. Actually, it is suggested that both p and q be larger than the maximum value of the range values being encrypted, so if we were just doing the ASCII values n should be larger than 127 * 127 or 16129.

The value of n is simply the result of multiplying p times q.

$n = p * q$

To create e and d, we first must create a value φ which is:

$\varphi = (p-1) * (q-1)$

The value of e some arbitrary number that is less than n and relatively prime to φ. There should be many numbers that fit this criteria and any one is valid. A number x is relatively prime of y if they have no divisor in common other than 1 (the term "co-prime" is also used to refer to this relationship). This means the GCD (Greatest Common Divisor) is 1. The GCD of two numbers can be computed via the Euclidean Algorithm.

The value of d can be calculated once e has been determined. The value of d is the inverse of e modulo φ. Which means

$(e * d) \bmod = 1$

Or for us computationally minded people:

$d = (1 + k\varphi) / e$ ➔ for some integer value k, d is the quotient of $(1 + k\varphi) / e$ when $(1 + k\varphi)$ can be evenly divided by e.

The calculation of both e and d is done via trial and error. You start with a possible value and see if it is valid. If not, you change the value and try again. This is done in a loop which exits when a valid value is found. Some algorithms will use the Xth valid value to try to add a bit more security to the values.

Algorithmic Help for the Large Exponentiation

The following mathematic algorithm will help reduce the size of the numeric values being used. This allows us to only deal with huge numbers instead of ridiculously gigantic numbers.

```
// The following code is equivalent to C = M^e mod n
C = 1;
for ( int i = 0 ; i < e ; i++ )
    C = ( C * M ) % n;
```

Key Creation and Storage

When creating the public and private keys, your program is to allow the user to either provide his/her own prime numbers or have the program randomly generate some prime numbers.

If the user will provide the prime numbers, prompt the user for each prime number and verify that the numbers are actually prime. There are algorithms that can be found via a google search that will verify if a number is prime. Research the internet or a numerical analysis book and INCLUDE A REFERNCE to what you found in your README file. Failure to do this will result in a lower score on this project. Please verify that the program/code used actually does work.

If the user wants the program to generate the prime numbers, your program can randomly select the prime numbers from a file that contains at least 20 prime numbers. This file should contain one prime number per line of the file. You are to create this file with appropriate prime numbers, but expect that the user of your program may replace this file with one of their own. Thus the name and location of this file need to be included in the README file for the program. This type of supplemental file is sometime referred to as a "resource file" so a name like "primeNumbers.rsc" is suggested.

The public and private keys that are to be created and used by your program are to be stored in files using the XML format. Note that the names of the XML tags must match what is given below, but the spacing need not. Also, the order of the e and n values in the public key and the order of the d and n values in the private key can be reversed. The part of your program that uses the public and private keys must be able to read the key values from any XML formatted file using the proper XML tag names. The names of the files themselves are left up to you. It is suggested that the user be allowed to assign a name or a prefix to the name of the keys.

The public key is to be store as:

```
<rsakey>
   <evalue>value-of-e</evalue>
   <nvalue>value-of-n</nvalue>
</rsakey>
```

The private key is to be stored as:

```
<rsakey>
   <dvalue>value-of-d</dvalue>
   <nvalue>value-of-n</nvalue>
</rsakey>
```

Examples of such file can be found at:

- Public Key Example
- Private Key Example

Message Blocking

The way a truly secure RSA program works is to encode the entire message into a single numeric value. For example, let us say the message is:

> Meet me outside SCE at 10pm.

We could translate this into a single numeric value by summing the ASCII numeric value for each character multiplied by 128^{pos} where pos is the character's position in the message. The decimal values for the first few and last few ASCII characters in the message are:

> M – 77 e – 101 m – 109 p – 113 t – 116 space – 32 0 – 48 1 – 49 . – 46

The value of the above message would be calculated as follows (note the message contains 28 characters):

$$77*128^0 + 101*128^1 + 101*128^2 + 116*128^3 + 32*128^4 + 109*128^5 + 101*128^6 + \ldots + 32*128^{22} + 49*128^{23} + 48*128^{24} + 113*128^{25} + 109*128^{26} + 46*128^{27}$$

The problem with the above idea is that the numeric value gets insanely large even when dealing with fairly short messages. So most often, RSA will "block" a message into smaller chunks and then encode each of those smaller chunks as separate numeric values. Note that industrial sized RSA programs may combine up to 100 characters into a single block. Our program will be using block sizes of 10 or less. We may try to push things to a block size of 15 or 16 to see if our program can handle them or not. Actually, if we properly write the huge-integer class, there should be nothing stopping us from using really big block sizes.

IMPORTANT NOTE: The prime numbers used for p & q when generating the keys MUST result in a n value that is larger than the maximum number for the message. If we were using a blocking size of 16 characters, the maximum number is somewhere up near $5x10^{33}$. This would require the prime numbers of p and q to be at the very least 17 decimal digits in length. For a blocking size of 8 characters, prime numbers of 8 or 9 decimal digits in length would be needed since the maximum number is up near $7x10^{16}$.

Some big prime numbers can be found at:

- **http://www.bigprimes.net/archive/prime/14000000/**
- **http://primes.utm.edu/lists/small/millions/**

The input for the blocking portion of your program will be the name of a file and a block size. The file will be the ASCII file containing the original message that is to be encoded using the RSA algorithm. Your program should then create a new file that contains the numeric value of each block on its own line. These numeric values should be written as decimal numbers. If the number of characters in the original message file is not a multiple of the block size, assume there are additional NULL characters (ASCII value 0) to fill in the last block. Your program is allow the used to name this "blocked" file

Note that this blocking operation will also need to be performed in reverse as the final step when decrypting a message. This reverse operation will need to read in a number from each line in the file and break this down into B ASCII characters, where B is the blocking size. Since the NULL character (ASCII value 0) should never be part of the original file and are only used to pad in the last block, if a NULL character is "unblocked", don't write it out to the final decoded file.

To help increase the readability of the blocked file, we will only work with a subset of the ASCII character set. These will include the 95 printable ASCII characters from space to tilde (ASCII decimal values of 32 to 126) and the following 5 special characters for a total of 100 characters.

- NULL character – ASCII decimal value 0, Escape character \0
- Vertical Tab – ASCII decimal value 11, Escape character \v
- Horizontal Tab – ASCII decimal value 9, Escape character \t
- New Line – ASCII decimal value 10, Escape character \n
- Carriage Return – ASCII decimal value 13, Escape character \r

By having only 100 characters in our alphabet, every 2 digits from the numbers in the blocked file will correspond to an ASCII character from our original message. The characters and their blocked numeric values can be determined by the following table. The left most column gives the ten's digit of the character while the top most row gives the one's digit of the character. Thus the character 'H' will be represented in the blocked file with the number of 45.

	0	1	2	3	4	5	6	7	8	9
0	\0	\v	\t	\n	\r	SP	!	"	#	$
10	%	&	'	(	)	*	+	,	-	.
20	/	0	1	2	3	4	5	6	7	8
30	9	:	;	<	=	>	?	@	A	B
40	C	D	E	F	G	H	I	J	K	L
50	M	N	O	P	Q	R	S	T	U	V
60	W	X	Y	Z	[	\	]	^	_	`
70	a	b	c	d	e	f	g	h	i	j
80	k	l	m	n	o	p	q	r	s	t
90	u	v	w	x	y	z	{	|	}	~

For the printable ASCII characters, their numeric value from the above table corresponds to the characters ASCII decimal number – 27. Note that ASCII character 'H' has ASCII decimal number of 72 which is 27 more than 45. Similarly ASCII character 'd' has ASCII decimal number of 100 which is 27 more than 73.

Using this scheme the message **Meet me outside SCE at 10pm.** become:

$50*100^0 + 74*100^1 + 74*100^2 + 89*100^3 + 5*100^4 + 82*100^5 + 74*100^6 + \ldots +$
$5*100^{22} + 22*100^{23} + 21*100^{24} + 85*100^{25} + 82*100^{26} + 29*100^{27}$

or

29828521220589700542405605747378888990840574820589747450

If we used a blocking size of 8 (8 characters per number) we get the following:

0574820589747450
0574737888899084
2205897005424056
0000000029828521

Note that the last number is padded with 4 null characters, which we can really ignore since integer values always ignore leading zero anyway.

Encryption/Decryption

You should note that the encryption process and the decryption process is exactly the same. Both are done by the use of the formula:

$$\text{OutputNumber} = (\text{InputNumber})^{exp} \bmod n$$

The difference between encryption and decryption occurs depending on which key you are using. If the key file contains a public key pair of (e, n), the exp specified in the above formula will be the e value. If the key file contains a private key pair of (d, n), the exp specified in the above formula will be the d value. The InputNumber will be the value on a line in a "blocked file" and the OutputNumber will become the value on a corresponding line in another "blocked file".

When running this part of the program, you should prompt the user for the name of a blocked file to be used for the InputNumbers, the name of a key file that will contain either a public or private key pair, and the name of file to create the blocked file containing the OutputNumbers.

Note that it is the responsibility of the user of the program to provide the correct key at the correct time.

One should also note that RSA can be used for restricted delivery, message signing or both. These depend on which key (or keys) is used during the encryption process and the decryption process. If the user gets this confused, the decrypted message won't be readable (which is the same as making sure the message doesn't get read by the wrong people).

Typically the overall process for restricted delivery occurs as follows.

1. Person 1 creates a public-private key set.
2. Person 1 sends the public key to Person 2 but keeps the private key
3. Person 2 creates an ASCII text message.
4. Person 2 blocks the message into a blocked file with block size X
5. Person 2 uses Person 1's public key to encrypt the blocked file.
6. Person 2 send the encrypted blocked file to Person 1
7. Person 1 decrypts the encrypted blocked file using Person 1's private key to (hopefully) get the original blocked file Person 2 created in step 4. (Since Person 1 should be the only person with this private key, Person 1 should be the only person who can decrypt the message.)
8. Person 1 unblocks the file created in step 7 to (hopefully) get the original ASCII text message.

Program Input and Output

The first menu for your program is to allow the user to select one of the following four options:

1. **Key Creation** – Create the private key file and public key file from the two prime numbers of p and q.
2. **Block a file** – Create a "blocked file" from an ASCII text file and a block size value
3. **Unblock a file** – Create an ASCII text file from a "blocked file".
4. **Encrypt/Decrypt** – Create a new "blocked file" that is the result of running the encryption algorithm on each line of another blocked file using the key pair contained in a file specified by the user.

Programming Style

Your program must be written in good programming style. This includes (but is not limited to) meaningful identifier names, a file header at the beginning of each source code file, a function header at the beginning of the function, proper use of blank lines and indentation to aide in the reading of your code, explanatory "value-added" in-line comments, etc.

The work you turn in must be 100% your own. You are not allowed to share code with any other person (inside this class or not). You may discuss the project with other persons; however, you may not show any code you write to another person nor may you look at any other person's written code.

Project Submission

You are to submit this project using the assignment link in Blackboard for project 3.