> Concern for man himself and his fate must always form the chief interest of all technical endeavors, concern for the great unsolved problems of the organization of labor and the distribution of goods —in order that the creations of our mind shall be a blessing and not a curse on mankind. Never forget this in the midst of your diagrams and equations.
>
> Albert Einstein (*What I Believe*, 1930)

**Due.** At the beginning of lecture on Tuesday, 27 October. You may turn it in before that date in the Carpenter consulting room. Do not turn it in at Carpenter on the due date.

**Working with another person.** You may do this assignment with one other person. However, you must share the work equally; you should write the code together, type it in together, and debug it together. It is not okay for one person to write and debug one method and another person to write and debug another; that is not togetherness. Hand in only one assignment with both names on it.

**Note.** The first comment of your program must contain the name, Cornell ID, section number, section day, section time, and section instructor of each person who is to receive a grade for the assignment you hand in. Don't write it in by hand; type it in as the first comment of the file that contains method *main*.

**Grading.** You are going to write the bodies of four methods. Each method is worth 2 points for each: 1 point for correctness and 1 point for organization and style. For style, we expect good definition-comments on variable declarations and a loop invariant for each loop.

# 1 Synopsis

The purpose of this assignment is (1) to give you practice with programs that use arrays and (2) to introduce you to the topic of cryptosystems, which are used to provide privacy when sending messages over the internet.

A cryptosystem is a method of encoding messages so that only certain people can read them, by decoding them. Cryptography is the art of designing cryptosystems, and cryptoanalysis is the art of breaking them. In this assignment, you will write methods that, together with methods we provide, allow you to encode (or encrypt) and decode (or decrypt) messages using a method that cryptoanalysts have not been able to break. The method is called RSA, after its inventors, computer scientists Rivest, Shamir, and Adleman. RSA has gained a kind of celebrity status to the extent that some governments fear that their best efforts won't break the system.

Appended to this handout is a 1994 NY Times article about RSA. The article does a good job of telling you a bit about the current history on cryptosystems and the difficult social issues concerning cryptosystems that are in the air.

In general, there are two types of cryptosystems: secret-key systems and public-key systems.

In a secret-key system, both the sender and the receiver of a message use (and therefore must know) the same secret key to encode or decode a message. Thus, there is a potential problem in arranging for both parties to use the same key. Unless they meet in person, under guarded conditions, someone might intercept the secret key and be able not only to decode messages but also to forge fake messages from either party to the other.

A public-key system avoids this problem. In such a system, two keys are used: A *public* key is used to encode a message, and a *private* key is used to decode it.

If Prof. Gries wants people to send him messages that others can't read, he first generates a (public key, private key) pair; he then makes the public key available to everyone; people then use the public key to encode messages to send to Prof. Gries. Only Prof. Gries —and not even the sender— can decode a message, since Prof. Gries is the only one with the private key.

RSA is a public-key system. In this assignment, you will learn the basics of RSA. Of course, we won't send encoded messages over the internet (although we could), but you will understand the coding-encoding process.

# 2 The assignment

## 2.1 Overview

Each following subsection contains some information and possibly instructions for dealing with part of the assignment. You may want to scan through the whole assignment before starting, to get an idea about what you are being asked to do. However, when actually doing the assignment, study the sections in the order given, and finish the part of the assignment dealing with one section before proceeding to the next section. In this way, gradually the whole idea of the RSA algorithm and its coding will be revealed to you, in what we hope is a simple manner.

Decryption and encryption computations will be done using type **long**. A **long** integer lies in the range −9223372036854775808 .. 9223372036854775807. Be sure that any variables that you declare in order to store temporary results of a computation are declared with type **long**. Note that other variables, for example one that contains the index of a character in a String, can have type **int**.

All the functions that you have to write appear in class *Crypto*, which you can get in the usual manner from the CS100A home page. These functions already have complete specifications (as comments). In addition, you will want to get a copy of method *main* from the home page, which you will use to generate the output that is to be handed in.

## 2.2 Public and private keys

In RSA, each public or private key is actually a pair $(k, m)$ of integers, where $k$ is called the *key* and $m$ the *modulus*. The public and private keys use the same modulus, so we can think of the public-private pair of keys as

public: $(puk, m)$
private: $(prk, m)$

To send Prof. Cardie an encoded message, you encrypt the message using her public key $(puk, m)$ and then send her the encrypted message. Provided she has kept her private key $(prk, m)$ to herself, only Prof. Cardie can decrypt and read the message.

The generation of public-private key pairs is outside the scope of this homework —see the attached article for a bit of information on it. So, for this homework, we will use the following public-private key pairs — note that for the last pair, we give you only the public key, so that you can encrypt messages that only WE (who know the private key) can read.

| puk | prk | m |
|---|---|---|
| 401 | 137 | 551 |
| 229 | 349 | 399 |
| 241 | 481 | 551 |
| 109 | | 493 |

All the integers in these keys are rather small. If it were known that all the integers in the keys were less than 1,000 (say), one could simply try all possibilities in order to decode a message. However, since we are using **long** integers, integers in keys can be up to 9223372036854775807. Now the idea of testing all possibilities becomes infeasible. And it is even possible to allow larger integers in the private-public key pairs, by using other representations of the integers. Thus, the sheer number of possibilities for keys helps make RSA feasible as a cryptosystem.

## 2.3 Values of type char

A **char** value is an integer in the range 0..65535. Furthermore, the characters that we actually use are in the range 0..255 (the other characters are for different languages and other things). To see this, perform the following tests. First, execute a program whose method *main* has the following body:

```
int i= 0;
// Invariant: The characters repre-
// sented by the integers 0..i-1
// have been printed, 25 to a line.
while (i != 255) {
    if (i%25 == 0)
        System.out.println();
    System.out.print((char)i);
    i= i+1;
    }
```

This prints the character represented by each integer in the range 0..255. Note the *conversion* "(**char**)i", which says to treat **int** value $i$ as a **char**.

As a second test, execute a program whose method *main* has the following body.

```
String s= ''abcxyz ABCXYZ 012789'';
int i= 0;
// Inv: The characters in s[0..-i]
// have been printed one to a line,
// along with their integer representa-
// tions
while (i != s.length()) {
    char c= s.charAt(i);
    System.out.println(c + ''  '' +
                       (int) c);
    i= i+1;
    }
```

Execution will print the characters in $s$ together with their representations. Note the conversion "(**int**) c", so that the character $c$ is treated as an integer.

In summary, the characters usually used in Java programs are represented by integers in the range 0..255; (**char**) $i$ is used to convert an integer $i$ to the character that $i$ represents, and (**int**) $c$ is used to convert a character to the integer that represents it.

## 2.4 Arithmetic modulo $m$

Let $m > 0$ be an integer. For any integer $i$ (which could be negative), $mod(i, m)$ is the integer $r$ that satisfies:

$$i = q * m + r \quad \text{and} \quad 0 \le r < m \quad \text{(for some } q\text{)}$$

Function $mod(i, m)$ is related to the Java "remainder" operation $i \; \% \; m$, but they differ when $i$ is negative. For example, $mod(-7, 5) = 3$, but $-7\%5 = -2$.

In doing mathematics, we often write $mod(i, m)$ as an infix operation:

$$i \ \mathbf{mod}\ m \quad .$$

However, in Java, we have to write $mod(i, m)$, since *mod* is a method (which happens to be a function). We have written function *mod*; it appears in class *Crypto*. Study this function, for you will be responsible for understanding the difference between *mod* and %.

To "reduce a value $v$ (say) modulo $m$" means to produce the value $mod(v, m)$ (and use that value in place of $v$). For example, to reduce 30 modulo 7 is to produce the value 2.

"Performing an integer computation modulo $m$" is done by reducing each result of a negation, addition, subtraction, multiplication, or exponentiation modulo $m$. Thus, all values in the computation are kept in the range $0..m-1$, using function *mod*.

In RSA, all computations are performed using modulo $m$ arithmetic, where $m$ is the modulus for the public and private keys.

Function $Crypto.exp(a, b)$, for $b > 0$, yields the value $a^b$, which is $a$ multiplied by itself $b$ times. For example, $Crypto.exp(-2, 3) = -8$.

As your first programming task, we ask you to fill in the body of function $Crypto.expMod(a, b, m)$, which performs exponentiation modulo $m$. The simplest way to do this is to copy the body of *Crypto.exp* to the body of *Crypto.expMod* and then to modify the body to perform all computations on **long** values modulo $m$. Remember: each negation, addition, and multiplication should be performed modulo $m$. (However, the computations involving $y$ alone need not be changed, since $y$ is always less than $m$.) Be sure to change any comments, like the loop invariant, appropriately.

Hints: Test this code before proceeding by modifying method *main* in the CUCS Java Application stationery.. In order to compile Crypto.java without errors, you will have to *temporarily* comment out methods *encrypt*, *fancyEncrypt*, and *fancyDecrypt*.

## 2.5 Encryption and decryption functions

Suppose we have a key $(p, m)$ —it doesn't matter whether it is public or private. In the RSA algorithm, an integer $i$ is transformed (encrypted or decrypted) to an integer $j$ using the formula

$$j = i^p \ \mathbf{mod}\ m \quad .$$

Because of the way RSA keys are generated, the following holds for a public key $(puk, m)$ and associated private key $(prk, m)$:

$$(i^{puk} \ \mathbf{mod}\ m)^{prk} \ \mathbf{mod}\ m = i$$

This means you can encrypt an integer with the public key and decrypt it with the private key, as we asserted earlier. But note that encryption and decryption of an integer use the same algorithm: exponentiation mod $m$. In the previous subsection, you were asked to write this algorithm.

To encrypt a message given in a String $s$, we encrypt each of its characters —viewing them as integers. This yields as the encrypted message an array of integers.

Decryption is the inverse: given an array of integers that represents an encrypted message, perform the decryption algorithm on each of the integers to determine the corresponding character.

We have written the decryption function for you, as method $Crypto.decrypt(c, k, m)$. Before you read any further, you should test it using the statements

$$long[\,]c = \{383, 277, 197, 98, 98\};$$
$$System.out.println(Crypto.decrypt(c, 137, 551));$$

in your method *main*. "CS100" should be printed out by execution of these two statements.

Look carefully at the statement

$$s = s + (char)expMod(c[i], k, m);$$

in *Crypto.decrypt*. Here, "(char)" is an indication that the integer should be treated as a character. Without "(char)", the integer would be treated as an integer. To see the difference, delete "(char)" and execute the program again and notice that the integer representations of the characters are printed. After looking at this output, don't forget to reinsert "(char)" in the statement.

Your task is to write the body of function *Crypto.-encrypt*. This is much like *Crypto.decrypt*, but its input is a String instead of an array and its output is an array instead of a String. Check this function out carefully before proceeding. Note that

$$Crypto.encrypt(\text{“}CS100'', 401, 551)$$

should produce the array $\{383, 277, 197, 98, 98\}$.

## 2.6 More secure encryption

The simple encryption used in the previous section is not as secure as it seems. Encoding each character separately simply provides another representation of each character, which is not much different than encoding each character by its successor in the alphabet (e.g.. encoding "New" as "Ofx"), and the frequency of characters and their juxtaposition can often give useful clues to decoding the message. Further, the longer a message, the easier the decoding gets.

So, we ask you to fill in the bodies of two fancier encryption-decryption methods:

*Crypto.fancyEncrypt*
*Crypto.fancyDecrypt* .

In *fancyEncrypt*, each element $c[i]$ of the encrypted message depends in a complicated way on its predecessors $c[0..i-1]$. First, we have

$$c[0] = \text{the RSA encryption of character } s[0]$$

where $s$ is the string to be encrypted. Second, each other element $c[i]$, in turn, is the RSA encryption of the number

$$(s[i] + c[i-1]) \textbf{ mod } m \quad .$$

Thus, $c[i]$ depends indirectly on the characters $s[0..i-1]$, not only on $s[i]$. Now, decryption is almost impossible, even if one knows the algorithm for encrypting and decrypting, unless one knows the private key.

Your first task for this subsection of the assignment is to write and check out the body of *fancyEncrypt*. In debugging this method, use the fact that

$$Crypto.fancyEncrypt(\text{“}aaaa'', 401, 551)$$

should produce the array $\{279, 173, 93, 285\}$. Note that in this encrypted message, there is no way to recognize that the message consists of the same character repeated four times.

Your second task for this subsection is to write and check out the body of method *fancyDecrypt*. This will take some thought. Be sure to add full specifications for the function in the comments that precede it.

## 2.7 What to hand in

Please hand in a copy of class *Crypto*, showing the methods that you wrote:

*expMod*,
*encrypt*,
*fancyEncrypt*, and
*fancyDecrypt*.

Secondly, hand in a copy of an execution of the following method *main*, using the CUCS Java Application stationery. (you can get method *main* from the CS100A home page):

```
public static void main(String args[]) {
    // initialize Text object in to read
    // from standard input.
    TokenReader in = new TokenReader(System.in);

    int i;
    long [] c;

    System.out.println(``one ``);
    c= Crypto.encrypt(``one ``,401,551);
    // Print array c
        for (i= 0; i != c.length; i= i+1)
            System.out.print(`` '' + c[i]);
        System.out.println();
    System.out.println(
           Crypto.decrypt(c,137,551));

    System.out.println(``t wo'');
    c= Crypto.fancyEncrypt(``t wo'',229,399);
    // Print array c
        for (i= 0; i != c.length; i= i+1)
            System.out.print(`` '' + c[i]);
        System.out.println();
    System.out.println(
```

```
        Crypto.fancyDecrypt(c,349,399));

    System.out.println(``three'');
    c=Crypto.fancyEncrypt(``three'',109,493);
    // Print array c
        for (i= 0; i != c.length; i= i+1)
            System.out.print(`` '' + c[i]);
        System.out.println();
    }
```

Note that you cannot tell whether you have encrypted the message "three" correctly, because you cannot decrypt it —only we have the private key. Hence, you should be sure that all the methods you have written are correct before executing this main program for the final time.

For your own interest, you and a friend might try sending messages back and forth over email. Agree on one of our public-private key pairs. Encode a message and email the resulting sequence of integers to the other person for decoding. You should be able to cut-and-paste, so that neither of you has to type in a long set of numbers. For example, print out an encoded message as a sequence of integers separated by commas, perhaps 10 to a line, e.g.

3, 303, 195, 3, 83, 97, 100, 101, 75, 95,
58, 45

Then copy-and-paste these into an email message and send them. On the receiving end, copy-and-paste the integers into a **long** array declaration within a Java program:

**long** [] c = {
3, 303, 195, 3, 83, 97, 100, 101, 75, 95,
58, 45
};

and execute some code to decrypt the message.

If you wish, send cardie@cs.cornell.edu a message that is encoded using the public key (109, 493); she will email you back the corresponding decoded message, so that you see that we really do have the private key.