Data Encryption and Decryption

Submission Date: 22th of March 11:59 PM

Contents

1 Introduction

Submit a **zipped folder** that is **only** containing your Java source files (*.java) in course's Black Board.

Please use the following naming convention for the submitted folders:

YourPSLetter_CourseCode_Surname_Name_HWNumber_Semester
Example folder names:

- **PSA_COMP130_Surname_Name_HW2_S19**
- **PSB_COMP130_Surname_Name_HW2_S19**

Additional notes:

- Using the naming convention properly is important, **failing** to do so may be **penalized**.
- **Do not** use Turkish characters when naming files or folders.
- Submissions with unidentifiable names will be **disregarded** completely. (ex. "homework1", "project" etc.)
- Please **write your name** into the Java source file where it is asked for. **Failing** to do so may be **penalized**.
- If you are resubmitting to update your solution, simply append **v#** where # denotes the resubmission version. (i.e. **v2**)

1.1 Academic Honesty

Koç University's *Statement on Academic Honesty* holds for all the homework given in this course. Failing to comply with the statement will be penalized accordingly. If you are unsure whether your action violates the code of conduct, please consult with your instructor.

1.2 Aim of the Homework

This homework aims to give you the chance to practice design and implementation of methods using control statement and expressions. You are asked to write a console program where you will be implementing a fundamental and widely used security algorithm called, **RSA (Rivest-Shavir-Adleman)** algorithm which was invented in 1977. The RSA algorithm make use of mathematical theorems and applications such as *Number Theory* and *Modular Arithmetic*. In the following sections, we will explain the details of the algorithm and you will be implementing the algorithm and the tools that it requires in your code.

1.3 Given Code

You are provided a code which has the necessary setup. It contains helper methods; variables and constants for you to start with.

1.3.1 Given Methods

You are given a helper method called **exptMod** with the following signature:

- **private int exptMod (int base, int power, int mod)**.

Please **do not modify this method**. You should call this method with the appropriate arguments for encrypting and decrypting a message.

In this program, you should implement the following methods which has the following signature:

- **void produceEncryptionKey ()** - The method used to create encryption key.
- **void produceDecryptionKey ()** - The method used to create decryption key.
- **void getUserMessage ()** - The method used to get 5-digit integer message.
- **void encryptDecryptMessage ()** - The method used encrypt and decrypt every digit of the original integer message.

Description of these methods are provided in the implementation section of this document.

1.3.2 Given Variables and Constants

Since we only learned so far the communication of methods through pass by value not by reference, we have provided you instance variables and constants in the code to allow communication of various methods. Please note that instance variables have a scope which covers the entire class. In other words, instance variables can be accessed from all the methods inside the same class. Please use these variables and constants in your code. Feel free to create additional ones if needed. Description of them are provided in the given code.

1.4 Further Questions

For further questions **about the project** you may send an email to **course SLs** at [comp198-spring-19-sl-group@ku.edu.tr] and **Ayca Tuzmen** at [atuzmen@ku.edu.tr]. Note that it may take up to 24 hours before you receive a response so please ask your questions **before** it is too late. No questions will be answered when there is **less than two days** left for the submission.

2 Part 1 - Implementation of RSA Algorithm

In this homework, you are asked to implement the RSA algorithm which is fundamental and widely used security algorithm. In summary, the algorithm consists of the following steps:

- Choose two distinct primes p and q of approximately equal size so that their product n = pq is of the required length.

- Compute φ(n) = (p-1)(q-1) (Euler's totient value).
- Choose a encryption/public key e, 1 < e < φ(n), which is coprime to φ(n), which means - gcd(e, φ(n))=1.
- Compute a decryption/private key d that satisfies the congruence e.d $\equiv$ 1 (mod φ(n)).
- Make the public key (n, e) available to others. Keep the private values d, p, q, and φ(n) secret.

Let's explain, how this algorithm can be implemented using the below tasks.

2.1 Step 1- Producing Primes and N Value

Creation of the encryption key, requires creation of:

- two prime numbers **(p and q)**
- n value (**n=p * q**)

In order to create the encryption key, we need to produce random two prime numbers, say *p* and *q*. Prime numbers are randomly generated and must satisfy the following conditions:

- p and q values should be between a lower and upper boundary (**lowerBound < p < upperBound) and (lowerBound < q < upperBound)**
- p and q should be two different numbers. **(p $\neq$ q)**
- n value of p, q should be less than the upper prime bound which is given as a constant value in the code. **(n < upperBound)**.

Task: Write a helper method which that takes no arguments and produces random numbers r $\in$ [lowerBound, lowerBound], and assign them to instance variables, *p and q*, with respect to given conditions.

Hint: You should implement a helper method to check whether a number is prime or not. To produce a prime number, generate random numbers until you find one which is prime.
Hint: You can use **RandomGenerator class** (a new instance of Random Generator class, say called rgen) to create random integer numbers.

Example: Lets say, we can randomly choose 3 and 5 for the primes p and q. The n value will be 15 (3*5).

2.2 Step 2 - Producing Euler's totient value

In number theory, Euler's totient function counts the positive integers up to a given integer n that are relatively prime to n. According to the RSA algorithm we should find the Euler's totient value of the n value.
Task: Write a helper method which that takes no arguments but produces Euler's totient of prime numbers p and q as such: φ(n) (**φ(n)= (p-1)*(q-1)**).
Example: Take p=3, q=5 and n=15, then 1, 2, 4, 7, 8, 11, 13, 14 are relatively

prime to n. Thus Euler's totient function returns 8. φ(n) = 8 (2*4) according to the above formulas.

2.3 Step 3 - Producing Encryption (Public) Key

The encryption key, *e*, can be any number that satisfies following conditions:

- Encryption key should be smaller than φ(n) variable.
- Greatest common divisor (GCD of φ(n) and the encryption key should be equal to 1. **(gcd (e , φ(n)) = 1**.
- Encryption key should be between a lower and higher bound **(LowerBound < e < UpperBound**) which are given as constant variables in the code.

In order to guarantee a number *e*, has a modular multiplicative inverse with respect to the specified modulo *m*, both numbers must be ***coprime*** or simply meaning the greatest common divisor of a number e and a modulo m) should be equal to 1, as shown below:

$$\boldsymbol{gcd\ (e\ ,\ m) = 1}$$

Task: You should write a method which takes two integers and returns ***greatest common divisor*** of them.

Important: You are **NOT allowed** to use any kind of ***Math library*** method in the implementation of the GCD method.

Task: You should implement the details of the given method with the following signature:

- **void produceEncryptionKey ()** - the method takes no arguments and returns nothing. It updates the instance variable called *encryptionKey* given the constraints for selecting an encryption key.

2.4 Step 4 - Producing Decryption (Private) Key

A decryption key is created using the inverse of the encryption key. To get the reverse of the encryption key we use the modular inverse of the encryption key. In order to do that, we use the operation named *modular multiplicative inverse*, and simply represented as such:

$$\boldsymbol{d.e \equiv 1\ (mod\ \varphi(n))}$$

The decryption key, *e*, can be any number that satisfies following conditions:

- Decryption key should be smaller than φ(n) variable.
- e. d ≡ 1. mod φ(n)

To compute the value for d, use the Extended Euclidean Algorithm to calculate d=e$^-$1 mod φ(n), also written d=(1/e)mod φ(n). This is known as modular inversion. Note that this is not integer division. The modular inverse d is defined as the integer value such that ed=1 mod φ(n).

2.4.1 Step 4.1- Create Modulo Operator

In computing, the modulo (remainder) operation finds the remainder after division of one number by another. Fortunately, Java has a quite simple remainder operator which will not work properly in arithmetic operations. When either a or n is negative, the naive definition of remainder method breaks down. For example, -27 % 5 = -2, when this should in fact be 2.

Since modular arithmetic operations returns non-negative results, you should upgrade the remainder method, we call it the **modulo method**.
Task: Write a helper method with the following signature:

- **int modulo (int number, int n)** - takes two integers as arguments, number **a** and **n** and returns **a (mod n)** in the range [0, n-1].

Be careful: If **a** is a negative number, **modulo** method should add **n** to the result of **a%n** until it is positive.

2.4.2 Step 4.2. - Create Multiplicative Inverse Method

The decryption key is obtained by finding the modular multiplicative inverse of the encryption key in modulo $\varphi(n)$

Task: Write a helper method **multInverse** with the following signature:

- **int multInverse (int e, int totient)** - takes two integers as arguments, **e** and $\varphi(n)$ and returns ***e^{-1} (mod n)***.

Hint: You should search for a decryption number, such that encryption key times decryption number is congruent to 1 in our modulo. Meaning, try to find a decryption key, such that the modulo of the multiplication of encryption and description keys in totient is equal to 1.

$$\boldsymbol{d.e \equiv 1 \ (mod\ \varphi(n))}$$

Example: As in the previous example, take φ(n) = 8, then we can take e to be 3. Then d can be chosen as 11.

2.5 Step 5 - Display of Encryption Key and Supporting Variables

When using RSA algorithm, we only share the public key, and don't share private keys, and other helper variables with others. However for the sake of practice, in this program we shall print the values of prime numbers, n, totient φ(n), and encryption and decryption keys on the console.

3 Part 2 - Get User Message

In this homework, we limit RSA algorithm to integer type messages only. The program shall prompt the user to enter a 5-digit integer type message. The

program shall check the size of the integer message and should keep on asking for a new message until a correct size message is entered.
Task: Implement the details of the given method with the following signature:

- **void getUserMessage ()** - the method takes no arguments and returns nothing. It updates the instance variables called *message* given the constraints for selecting an message.

Hint: You can write a helper method which checks the digit size of the message.
Example: You can use your KU ID, say 12345 as a message.

4 Part 3 - Encrypt and Decrypt Message

The program shall display the original message before encryption and after decryption.

4.1 Step 1 - Encrypting Single Digits

The program shall break the integer digit into single integer digits and encrypt every digit in the integer message using the same encryption key. It shall display the original integer digit and the encrypted digit on the console.

You should call the method given to you called **exptMod**, and pass digit to be encrypted, encryption key and n value.

exptMod method takes three arguments, called base, power, and mod. You should pass the value of single digit that you would like to encrypt to base value, encryption key to power and n value to mod values.

4.2 Step 2 - Decrypting Single Digits

The program shall decrypt every digit in the integer message using the same decryption key and display them on the console.

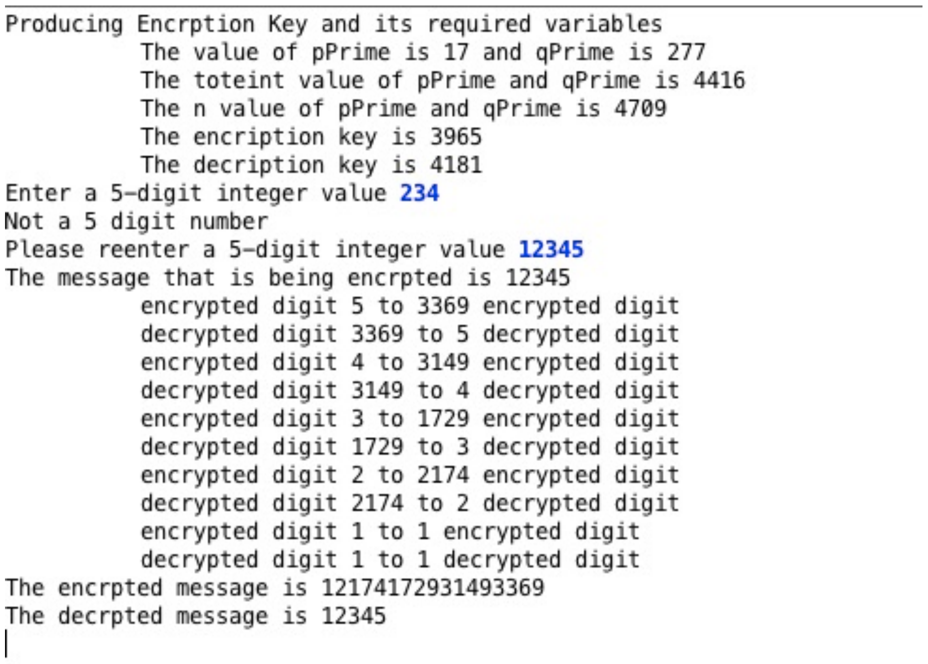
You should call the method given to you called **exptMod**, and pass digit to be decrypted to base variable, decryption key to power variable and n value to mod.
Hint: Note that, the **exptMod** method you will be using will make use of the **modulo** method that you will be implementing in Step 4.1 (secion 2.4.1).

4.3 Step 3 - Display Encrypted and Decrypted Messages

The program shall construct and display two string variables for storing the messages constructed using the encrypted and decrypted single digits. It shall put together the decrypted message back to 5-digit integer message and display on the console.

5 Sample Output

Below is a sample output of this program.



```
Producing Encrption Key and its required variables
          The value of pPrime is 17 and qPrime is 277
          The toteint value of pPrime and qPrime is 4416
          The n value of pPrime and qPrime is 4709
          The encription key is 3965
          The decription key is 4181
Enter a 5-digit integer value 234
Not a 5 digit number
Please reenter a 5-digit integer value 12345
The message that is being encrpted is 12345
          encrypted digit 5 to 3369 encrypted digit
          decrypted digit 3369 to 5 decrypted digit
          encrypted digit 4 to 3149 encrypted digit
          decrypted digit 3149 to 4 decrypted digit
          encrypted digit 3 to 1729 encrypted digit
          decrypted digit 1729 to 3 decrypted digit
          encrypted digit 2 to 2174 encrypted digit
          decrypted digit 2174 to 2 decrypted digit
          encrypted digit 1 to 1 encrypted digit
          decrypted digit 1 to 1 decrypted digit
The encrpted message is 12174172931493369
The decrpted message is 12345
|
```

Figure 1: Sample output

6 End of Project

Your project ends here. You may continue to tinker with the code to implement any desired features and discuss them with your section leader. **Do not** include any additional features that you implement after this point in to your submission.