Project Assignment - Part 2

Introduction

This is the second part of the project assignment in Networks and Security - Fall 2014.

This is an individual assignment, and must be passed, in order to attend the exam. As a prerequisite for the exam, it is to be considered a part of the exam, and rules for cheating are the same as the exam.

This time, you must create a program, and answer a few questions along the way.

Your solution must be uploaded in the e-learn.sdu.dk (Blackboard) system no later than **Wednesday December 17th. at 16.15.** Late submissions will not be accepted. Read later in this assignment the format of the hand-in.

The task

In this part, you must implement RSA, including generating keys, encryption and decryption. A template project can be downloaded from `http://imada.sdu.dk/~jamik/dm543-14/material/rsa.zip`. As numbers for encryption can be very large, you will be working with the BigInteger class from java in this project.

Tests have been prepared, and you should use these, i.e. do test driven development. In order to run the tests, use the gradle wrapper. Use the following command:

```
./gradlew build
```

If you are using windows, use the `gradlew.bat` file

All tests will at the beginning fail, as the code is not implemented, but as you progress in the assignment, they should pass. The tests are written in the Spock test framework, but knowledge of this is not a requirement.

Alphabet

Here is the following alphabet conversion:

```
0 = space, 1 = a, 2 = b, ..., 26 = z
```

We will use only 27 characters for our messages (26 lower case letters and the space). If you find other characters in the plain text message (input mes- sage) such as punctuation, white space, etc., simply treat the characters as spaces and assign them the value of 0.

Character Conversion

Before you can implement RSA, you must encode an input message as a number. We will use the following scheme to convert plaintext (input) messages to numbers:

1. Convert any extra characters to spaces as specified above.

2. Group the plaintext into sets of five characters per group. If the last grouping does not have exactly five characters, then append some space to the end of the plaintext message to fill out the last grouping. Each group must have five characters.
3. Convert each group into a separate number: If the grouping is $[c_4c_3c_2c_1c_0]$ then the number is

$$\sum_{i=0}^{4} c_i \cdot 27^i$$

Here is an example. Assume our plaintext grouping is [this]. First we translate the characters into numbers:

$$\begin{aligned} c_4 &= \text{'t'} = 20 \\ c_3 &= \text{'h'} = 8 \\ c_2 &= \text{'i'} = 9 \\ c_1 &= \text{'s'} = 19 \\ c_0 &= \text{' '} = 0 \end{aligned}$$

Then we compute the plaintext number:

$$20 \cdot 27^4 + 8 \cdot 27^3 + 9 \cdot 27^2 + 19 \cdot 27^1 + 0 = 10,793,358.$$

A decoding operation should be performed in the same way except that the process is reversed. A number is converted to its character grouping by using mod and div.

In the file *AlphabetConversion.java*, you find 4 methods you should implement. Use the description above, the java doc, and study the test file (*AlphabetConversionSpec.groovy*). Recommended order of implementation is:

1. `charToNumber`
2. `numberToChar`
3. `stringToNumber`
4. `numberToString`

Key Generation

In order to generate the public and private key, **e** must be chosen relatively prime to **z**, and **d** must satisfy $e \cdot d \bmod z = 1$. For both of these criterias, the extended euclidean algorithm can be used:

```
EXTENDEDEUCLIDEANALGORITHM(a, b)
 1  a_0 ← a
 2  b_0 ← b
 3  t_0 ← 0
 4  t ← 1
 5  s_0 ← 1
 6  s ← 0
 7  q ← ⌊a_0 / b_0⌋
 8  r ← a_0 − q · b_0
 9  while r > 0
10        do
11           temp ← t_0 − q · t
12           t_0 ← t
13           t ← temp
14           temp ← s_0 − q · s
15           s_0 ← s
16           s ← temp
17           a_0 ← b_0
18           b_0 ← r
19           q ← ⌊a_0 / b_0⌋
20           r ← a_0 − q · b_0
21  r ← b_0
22  return {r,s,t}
```

The return value provides the following: $r = \texttt{gcd}(a, b)$ and $s \cdot a + t \cdot b = r$

Implement the algorithm in the *NumberHelpers.java* class.

In the report: Provide an example, using the extended euclidean algorithm, shoving how e and d can be chosen.

Encryption and decryption

In order to do encryption and decryption, you should implement fast modular exponentiation. The algorithm is in pseudo code below, and the file is *NumberHelpers.java*.

```
RECURSIVEMODULAREXPONENTIATION(base, pow, modulus)
 1  if pow = 0
 2     then return 1
 3  if pow = 1
 4     then return base
 5  if pow mod 2 = 0
 6     then
 7           temp ← RECURSIVEMODULAREXPONENTIATION(base, pow/2, modulus)
 8           return temp · temp mod modulus
 9     else
10           temp ← RECURSIVEMODULAREXPONENTIATION(base, ⌊pow/2⌋, modulus)
11           return base · temp · temp mod modulus
```

You are now ready to implement encryption and decryption, using the tools you have developed.

Implement the rsa encryption and decryption in the *Rsa.java* file. Remember the encryption method must pad the input with spaces, if the length is not a multiple of 5.

In the report: Argue how large n at least must be, when you convert strings to numbers according to the scheme above. How large would n have to be, if the alphabet was size 53 (i.e. using large and small characters)?

Select a task

You must select one of the following two tasks

1: Configurable length

Make the length of the grouping a variable, so you can encode in blocks longer than 5. You must include extra tests to show this is working

You must also include a calculation/check if the primes chosen are large enough to cover the length of the blocks.

2: Chaining

This implementation has a serious flaw! Two identical blocks of letters will encrypt to the same ciphertext.

This task consist of designing a chaining scheme, so the encoding of a block will depend on the previous block.

This will then make sure that encrypting 'help help help help help' does not come up with 5 identical encrypted blocks.

Testing

Tests have been provided for the project, but you are free to implement further tests, if you find that something is not covered with a test.

What to turn in

You must make a zip file with the following structure:

```
<your-sdu-username>.zip
|_ code
|_ doc
```

- The *code* folder must contain the code, in the same structure as the handed out project.
- The doc folder must contain a single pdf with your report for this assignment.

In order to pass this assignment, your implementation should pass all tests, and hand in a clear and well-structured report.

The report

The questions in boxes must be answered, and you must provide a detailed example of encryption and decryption of a string of length at 10 characters. Your report should be at most 4 pages. You should not make index, frontpage or copy the assignment text.

1 Literature

1. Gradle: `http://gradle.org/`
2. Spock framework: `http://docs.spockframework.org/en/latest/`