CO3326 Computer Security 2015-16

Code specification CO3326 cw1

This description contains the requirements for the **code submission**, which has to be read in conjunction with the actual assignment you are already familiar with. It is important that you follow these instructions carefully, as your code submission will be processed automatically and will not be seen by the marker unless it passes all test cases that are detailed below.

General guidelines

- You will submit a *single JAR file* for each coursework following a *specific naming scheme*.
- Your code must compile and run against *Java 8*.
- The entry point in your code will have a main method, which receives one command-line argument, the absolute path to an *input file*.
- Your programme will read the input file and treats each line as a separate *test case* / separate instance of the problem.
- Each line in the test file is a JSON object.
- Your programme will print a single line of text, containing one JSON object, for every single input line. Therefore the number of input lines will match the number of output lines.
- The JSON objects, both input and output, have a *strict format* you will have to conform to and there are details provided below with examples.

Prerequisites

- Make sure you have **Java 8 Development Kit** installed.

```
    $ java -version
```

```
    java version "1.8.0_66"
    Java(TM) SE Runtime Environment (build 1.8.0_66-b17
)
    Java HotSpot(TM) 64-Bit Server VM (build 25.66-b17,
mixed mode)
```

- The minor version (in this case 66) is unimportant. If you don't have Java 8 Development Kit installed, install it: http://www.oracle.com/technetwork/java/javase/downloads/inde (available for all platforms).
- Set up the **Maven** build system. Instructions are available here: https://maven.apache.org/install.html (for all platforms).
- Double-check that you have at least Maven 3.3.3:

```
    $ mvn -v
```

```
    Apache Maven 3.3.3 (7994120775791599e205a5524ec3e0d
fe41d4a06; 2015-04-22T12:57:37+01:00)
    Maven home: /usr/local/Cellar/maven/3.3.3/libexec
    Java version: 1.8.0_66, vendor: Oracle Corporation
    Java home: /Library/Java/JavaVirtualMachines/jdk1.8
.0_66.jdk/Contents/Home/jre
    Default locale: en_US, platform encoding: UTF-8
    OS name: "mac os x", version: "10.11.2", arch: "x86
_64", family: "mac"
```

Setup

- Extract the provided ZIP file and you will have a **co3326** folder, which will be referred to as your *project folder*; navigate to it using a command-line prompt.
- Double-check that you have the following folder structure:

```
/co3326/
    |-- README.md
    |-- pom.xml
    |-- test.txt
    |-- /src/
        |-- /main/
            |-- /java/
                |-- /co3326/
                    |-- App.java
                    |-- Cw1.java
                    |-- Message.java
                    |-- User.java
            |-- /resources/
                |-- config.properties
```

- Edit the `pom.xml` file
- Look for the following lines:

```
<student.name>FirstnameLastname</student.name>
<student.srn>27644437</student.srn>
```

- Replace `FirstnameLastname` with your name using a *CamelCase* syntax, **without** blanks or dashes or underscores. Ex. if your name is Jane Smith, use *JaneSmith*.
- Replace `27644437` with your SRN.
- Save the `pom.xml` file.

Build

- Open a command-line prompt and navigate to your project folder (`co3326`).
- Build the project with the following command:

```
$ mvn clean package
```

- This should have an output ending with:

```
    ...

    [INFO] ------------------------------------------
---------------------------
    [INFO] BUILD SUCCESS
    [INFO] ------------------------------------------
---------------------------
    [INFO] Total time: 3.114 s
    [INFO] Finished at: 2016-01-06T00:28:35+00:00
    [INFO] Final Memory: 20M/171M
    [INFO] ------------------------------------------
---------------------------
```

- A new **target** folder should have appeared in your project folder (`co3326/target/`), which should contain - among others - a `FirstnameLastname-jar-with-dependencies.jar` file. Obviously, the file name will have your name in it, ex. if you're *Jane Smith*, the file will be `JaneSmith-jar-with-dependencies.jar`.

Test

- The JAR file obtained with the build process is executable and contains both the Java byte code and the source code.
- There is a *test file* in your project folder (`test.txt`), which is a valid test file and similar to the one that will be used by the automatic testing process when your submission will be evaluated, so this is the test file you will have to make sure that your code runs against successfully.
- In a command-line prompt issue the following cmmand:

```
    $ java -jar target/FirstnameLastname-jar-with-depen
dencies.jar test.txt
```

- Obviously, you'll use using the correct file name. The output should be:

```
    FirstnameLastname
    27644437
    {"communication":[{"text":"University of London"}]}
    {"alice":{},"communication":[{"text":"Hi Bob!"}]}
    {"alice":{},"bob":{},"charlie":{},"communication":[
{"text":"Hello World!"}]}
```

Develop

The source folder, where your code will have to go is `src/main/java/` within yout project folder. You have a **co3326** the top level package (`src/main/java/co3326`). Look at the `App.java` , `Cw1.java` , `Message.java` and `User.java` files which are already in there. These provide a starting point for your code and help you with the reading of the input file, parsing of the input lines and creation of JSON representations of the test cases as well as printing the JSON results. You can place your code next to these files. The *main* method is in the `co3326.App` class.

For serializing / de-serializing JSON objects, Google's Gson library is used (https://github.com/google/gson). The build system (i.e. Maven) sorts out the dependency for you and bundles the library in your JAR file when you build the project.

You may use an IDE of your choice. Eclipse, IntelliJ IDEA and NetBeans are popular choices and available for all platforms. Most IDEs, including the ones mentioned, have support for Maven and will recognise your project when you import it.

Required output: *Coursework 1*

Inspect the `test.txt` in your project folder:

```
{ "communication" : [ { "text" : "University of London"
} ] }
{ "alice" : { "rsa" : { "p" : 313, "q" : 787 } }, "comm
unication" : [ { "text" : "Hi Bob!" } ] }
{ "alice" : { "rsa" : { "p" : 313, "q" : 787 } }, "bob"
: { "rsa" : { "p" : 157, "q" : 641, "e" : 29203 } }, "c
harlie" : { "rsa" : { "p" : 373, "q" : 977, "e" : 25884
5 } }, "communication" : [ { "text" : "Hello World!" }
] }
```

The following output is correct for this test file, both in terms of *format* and *content*:

```
FirstnameLastname
27644437
{"alice":{"rsa":{"p":353,"q":739,"e":74609,"n":260867,"
r":259776,"d":250577}},"bob":{"rsa":{"p":37,"q":733,"e"
:17359,"n":27121,"r":26352,"d":1471}},"charlie":{"rsa":
{"p":313,"q":1009,"e":224249,"n":315817,"r":314496,"d":
305993}},"communication":[{"text":"University of London
```

","encoded":[85,110,105,118,101,114,115,105,116,121,32,111,102,32,76,111,110,100,111,110]},{"encoded":[51695,7876,216371,187465,245967,86567,170449,216371,84549,65399,158992,138625,51377,158992,272409,138625,7876,101075,138625,7876]},{"text":"University of London :: intercepted","encoded":[85,110,105,118,101,114,115,105,116,121,32,111,102,32,76,111,110,100,111,110,32,58,58,32,105,110,116,101,114,99,101,112,116,101,100]},{"encoded":[180276,96682,290466,200611,288925,151909,239338,290466,167979,71118,48758,12480,246717,48758,44604,12480,96682,278234,12480,96682,48758,49069,49069,48758,290466,96682,167979,288925,151909,186226,288925,70993,167979,288925,278234]},{"text":"University of London :: intercepted :: received","encoded":[85,110,105,118,101,114,115,105,116,121,32,111,102,32,76,111,110,100,111,110,32,58,58,32,105,110,116,101,114,99,101,112,116,101,100,32,58,58,32,114,101,99,101,105,118,101,100]}]}
{"alice":{"rsa":{"p":313,"q":787,"e":44429,"n":246331,"r":245232,"d":136037}},"bob":{"rsa":{"p":103,"q":521,"e":23657,"n":53663,"r":53040,"d":16553}},"charlie":{"rsa":{"p":439,"q":857,"e":341965,"n":376223,"r":374928,"d":95077}},"communication":[{"text":"Hi Bob!","encoded":[72,105,32,66,111,98,33]},{"encoded":[353251,34421,150268,286700,56706,306602,212817]},{"text":"Hi Bob! :: intercepted","encoded":[72,105,32,66,111,98,33,32,58,58,32,105,110,116,101,114,99,101,112,116,101,100]},{"encoded":[194469,350993,125350,116698,175587,292246,102367,125350,130545,130545,125350,350993,59464,136675,36166,298197,247651,36166,71422,136675,36166,265525]},{"text":"Hi Bob! :: intercepted :: received","encoded":[72,105,32,66,111,98,33,32,58,58,32,105,110,116,101,114,99,101,112,116,101,100,32,58,58,32,114,101,99,101,105,118,101,100]}]}
{"alice":{"rsa":{"p":313,"q":787,"e":118865,"n":246331,"r":245232,"d":243761}},"bob":{"rsa":{"p":157,"q":641,"e":29203,"n":100637,"r":99840,"d":48667}},"charlie":{"rsa":{"p":373,"q":977,"e":258845,"n":364421,"r":363072,"d":74933}},"communication":[{"text":"Hello World!","encoded":[72,101,108,108,111,32,87,111,114,108,100,33]},{"encoded":[291497,323875,261384,261384,322480,279160,194042,322480,327447,261384,353896,336665]},{"text":"Hello World! :: intercepted","encoded":[72,101,108,108,111,32,87,111,114,108,100,33,32,58,58,32,105,110,116,101,114,

99,101,112,116,101,100]},{"encoded":[294580,181265,1493
85,149385,283553,136995,216187,283553,197484,149385,200
153,290070,136995,251883,251883,136995,353286,60176,103
884,181265,197484,143653,181265,251364,103884,181265,20
0153]},{"text":"Hello World! :: intercepted :: received
","encoded":[72,101,108,108,111,32,87,111,114,108,100,3
3,32,58,58,32,105,110,116,101,114,99,101,112,116,101,10
0,32,58,58,32,114,101,99,101,105,118,101,100]}]}

The first two lines are *your name* (in CamelCase) and *your SRN*. The following 3 lines are the outputs corresponding to the 3 input lines. To explain the output, we'll look at the first two input-output pairs in turn.

First example

Input

This is the simplest input:

```
{ "communication" : [ { "text" : "University of London"
} ] }
```

Only the initial plain text is given: *University of London*. Nothing is given about **Alice**, **Bob** and **Charlie**, therefore *all* their key-pairs have to be generated by you.

Possible output

Outputs will vary depending on what *p*, *q* and *e* values you have generated for **Alice**, **Bob** and **Charlie**, but the correct format of the output is the following:

```
{
    "alice": { "rsa": { "p": 353, "q": 739, "e": 74609,
"n": 260867, "r": 259776, "d": 250577 } },
    "bob": { "rsa": { "p": 37, "q": 733, "e": 17359, "n
": 27121, "r": 26352, "d": 1471 } },
    "charlie": { "rsa": { "p": 313, "q": 1009, "e": 224
249, "n": 315817, "r": 314496, "d": 305993 } },
    "communication": [
        { "text": "University of London", "encoded": [8
5, 110, 105, 118, 101, 114, 115, 105, 116, 121, 32, 111
, 102, 32, 76, 111, 110, 100, 111, 110] },
        { "encoded": [51695, 7876, 216371, 187465, 2459
67, 86567, 170449, 216371, 84549, 65399, 158992, 138625
, 51377, 158992, 272409, 138625, 7876, 101075, 138625,
7876] },
        { "text": "University of London :: intercepted"
, "encoded": [85, 110, 105, 118, 101, 114, 115, 105, 11
6, 121, 32, 111, 102, 32, 76, 111, 110, 100, 111, 110,
32, 58, 58, 32, 105, 110, 116, 101, 114, 99, 101, 112,
116, 101, 100] },
        { "encoded": [180276, 96682, 290466, 200611, 28
8925, 151909, 239338, 290466, 167979, 71118, 48758, 124
80, 246717, 48758, 44604, 12480, 96682, 278234, 12480,
96682, 48758, 49069, 49069, 48758, 290466, 96682, 16797
9, 288925, 151909, 186226, 288925, 70993, 167979, 28892
5, 278234] },
        { "text": "University of London :: intercepted
:: received", "encoded": [85, 110, 105, 118, 101, 114,
115, 105, 116, 121, 32, 111, 102, 32, 76, 111, 110, 100
, 111, 110, 32, 58, 58, 32, 105, 110, 116, 101, 114, 99
, 101, 112, 116, 101, 100, 32, 58, 58, 32, 114, 101, 99
, 101, 105, 118, 101, 100] }
    ]
}
```


Important: the actual output will be a single line, but a pretty-print is used in this description such that you can better understand what is expected.

- The `rsa` field will be generated / computed entirely by you for **Alice**, **Bob** and **Charlie**.
- You will extract the initial message, encode it, push it back to the `communication` list and push all subsequent messages as the initial message gets sent, intercepted and received between

Alice, **Charlie** and **Bob**.

- For plain texts (non-encrypted), include the encoding; i.e. the message will have both the `text` and the `encoded` fields set.
- For encrypted texts, the encoded text suffices; i.e. the message will only have the `encoded` field set.
- The `communication` list will have to contain the trail of messages as the initial message travels from **Alice** to **Bob** and is intercepted by **Charlie**.
- **Alice** thinks she is communicating with **Bob**, but is in fact communicating with **Charlie**, who decodes **Alice**'s message, slightly alters it (appends *:: intercepted* to the original message, for example) and sends it to **Bob**. **Bob** will finally decode the message and mark it with *:: received* before sending it back to **Alice**.
- Each message is doubly-encrypted, when **A** sends a message to **B**, the message will be encrypted with **A**'s private key and **B**'s public key.
- Subsequently, when **B** receives the message, it will have to be doubly-decrypted, with **B**'s private key and with **A**'s public key.
- You may have to pay attention to the ordering of the double encryption and the double decryption.
- You may continue to simulate a message travelling back from **Bob** to **Alice** but only the first 5 messages will be checked by the automatic tester.
- More precisely, the `encoded` field of the 2nd message will be checked against the `encoded` field of the 1st message, using the *p*, *q* and *e* values you generated.
- Subsequently, it will be checked whether the decrypted 2nd message is a subset of the 3rd message, whether the 4th message is correctly generated from the 4th and whether the decrypted 4th message is a subset of the 5th message.

Second example

Input

```
{ "alice" : { "rsa" : { "p" : 313, "q" : 787 } }, "comm
unication" : [ { "text" : "Hi Bob!" } ] }
```

The initial plain text is given: *Hi Bob!*. In addition **Alice**'s *p* and *q* pair is also given; *e* will be generated by you. Nothing is said about **Bob** and **Charlie**, therefore *all* their key-pairs have to be generated by you.

Possible output

Outputs will vary depending on what *p*, *q* and *e* values you have generated for **Bob** and **Charlie** and what value have you picked for **Alice**'s *e*, but the correct format of the output is the following:

```
{
    "alice": { "rsa": { "p": 313, "q": 787, "e": 44429,
"n": 246331, "r": 245232, "d": 136037 } },
    "bob": { "rsa": { "p": 103, "q": 521, "e": 23657, "
n": 53663, "r": 53040, "d": 16553 } },
    "charlie": { "rsa": { "p": 439, "q": 857, "e": 3419
65, "n": 376223, "r": 374928, "d": 95077 } },
    "communication": [
        { "text": "Hi Bob!", "encoded": [72, 105, 32, 6
6, 111, 98, 33] },
        { "encoded": [353251, 34421, 150268, 286700, 56
706, 306602, 212817] },
        { "text": "Hi Bob! :: intercepted", "encoded":
[72, 105, 32, 66, 111, 98, 33, 32, 58, 58, 32, 105, 110
, 116, 101, 114, 99, 101, 112, 116, 101, 100] },
        { "encoded": [194469, 350993, 125350, 116698, 1
75587, 292246, 102367, 125350, 130545, 130545, 125350,
350993, 59464, 136675, 36166, 298197, 247651, 36166, 71
422, 136675, 36166, 265525] },
        { "text": "Hi Bob! :: intercepted :: received",
"encoded": [72, 105, 32, 66, 111, 98, 33, 32, 58, 58, 3
2, 105, 110, 116, 101, 114, 99, 101, 112, 116, 101, 100
, 32, 58, 58, 32, 114, 101, 99, 101, 105, 118, 101, 100
] }
    ]
}
```

Important: the actual output will be a single line, but a pretty-print is used in this description such that you can better understand what is expected.

- Again, you may continue to simulate a message travelling back from **Bob** to **Alice** but only the first 5 messages will be checked by the automatic tester, as explained above.

Help

- You may restrict *p* and *q* to be primes in the **(8, 1024)** interval.
- Pick *e* randomly such that it is co-prime with *r* and compute *d* accordingly.
- You can assume that the `communication` array initially always

contains the plain text of the message that **Alice** is to send to **Bob** that will be intercepted by **Charlie**.
- To *encode* plain texts (i.e. strings) to an array of integers that can be encrypted / decrypted, you may use the following functions (**Java 8**):

```
    public static int[] encode(String message) {
        ByteBuffer buffer = ByteBuffer.wrap(message.get
Bytes());
        return IntStream.generate(buffer::get).limit(bu
ffer.remaining()).toArray();
    }

    public static String decode(int[] message) {
        return Arrays.stream(message)
                .mapToObj(i -> (char) i)
                .reduce(new StringBuilder(), (sb, c) ->
sb.append(c), StringBuilder::append)
                .toString();
    }
```

- These functions use the ASCII values of the characters that make up the string to convert between string and a list of integers.

Submission

Once you're happy with your code

- rebuild your project, following the instructions in the Build section,
- re-test your project, following the instructions in the Test section,
- double-check that the output in the required format,
- submit the `FirstnameLastname-jar-with-dependencies.jar` JAR file (which will obviously have your name in it, ex. if you're Jane Smith, the file will be `JaneSmith-jar-with-dependencies.jar`), which is located in your project's `target` folder.

Important

Only a JAR file, which uses the correct naming scheme and produces output in the described format will be looked at, and **only** if it passes the automatic tests. A ZIPed project folder or individual Java or Class files will **not** be looked at, and will be awarded a *0*

mark. If you correctly follow the description above, the JAR file you submit will contain the source code that you worked on and will give the marker the opportunity to check it, once your byte code passes the automatic execution phase.

Last modified: Monday, 18 January 2016, 11:26 AM