National University of Singapore
School of Computing

CS2105 **Assignment 3** Semester 2 AY16/17

Submission Deadline

5 April 2017 (Wednesday), 6pm sharp. Do not leave your submission to the last minute in case of unforeseeable situation, e.g. network congestion. You have 99 chances of submissions and only the last submission will be graded.

2 points penalty will be imposed on late submission (Late submission refers to submission or re-submission after the deadline).

Learning Objective

In this assignment, you will have a taste of Java APIs for cryptography. This programming assignment is worth 6 marks and shall be completed **individually**.

Writing Your Programs

You are free to write your program on any platform/IDE that you are familiar with.

However, you are responsible to ensure that your program runs properly on **`sunfire`** because **we will test and grade your program on `sunfire`.**

Program Submission

Please submit your program to **`CodeCrunch`**: https://codecrunch.comp.nus.edu.sg. You just need to submit **`Alice.java`**. We will use our own **`Bob.java`** (which is the same as the one given to you) for testing. Note that **`CodeCrunch`** is used only for program submission and no test case has been mounted on it.

Grading

Your programs will be graded according to their correctness using a grading script:

- **[2 points]** **`Alice.java`** compiles on **`sunfire`** with errors and follows the specified **`Java`** command to launch (see section below).
- **[4 points]** **`Alice.java`** correctly decrypts messages and saves them to a file. During grading we do not care about messages that your program prints on the screen. It just checks if the generated file is identical to the one the sender (Bob) has.

Introduction

Bob wants to send a confidential document to Alice. Here is how they will do it.

1. Alice keeps a copy of Bob's RSA public key. She generates an AES session key and encrypts it with Bob's public key. Alice then sends this encrypted session key to Bob. Bob uses his RSA private key to decrypt and get the session key.
2. Bob sends the confidential document (a text file) to Alice over **TCP** using the following algorithm:

```
while the file has more lines to read do
    read a line and encrypt it with AES session key;
    send encrypted message as a SealedObject to Alice;
end
```

3. Alice receives the encrypted messages, decrypts them and saves the plaintext to a file.

Skeleton Programs

`Bob.java` is complete and given to you. Your task in this assignment is to complete the given skeleton program **`Alice.java`** so that Alice can (1) successfully generate and send session key to Bob, (2) receive all the messages from Bob, decrypt and save them to a file. The file Alice generates must be identical to the confidential document Bob keeps.

`Bob.java`

The **`Bob.java`** program is complete and should not be changed. To run it, type:

```
java Bob <BobPort>
```

For example:

```
java Bob 9000
```

listens at port 9000 and waits for Alice to connect.

Upon startup, Bob will read his RSA public and private keys from the files **`public.key`** and **`private.key`** stored in the same directory as **`Bob.java`**.

Bob runs a TCP server. Once connected by Alice, Bob first receives an encrypted session key from Alice and decrypts it with his RSA private key. He then sends the file **`docs.txt`** line by line to Alice, each line encrypted with the session key.

You should read **`Bob.java`** carefully because most of the codes you are supposed to write in **`Alice.java`** have counterparts (i.e. reference) in **`Bob.java`**.

> You will find two txt files in the **"**bob" folder. **If you use Windows**, please replace docs.txt with docs_windows.txt.
> Windows uses a different newline ("\r\n") compared to all other platform ("\n"). If you do not use docs_windows.txt, your received messages might not be equivalent to the ones sent by Bob due to different newline encodings.
> If you use any other operating system than Windows, please use the original file.
> During grading we will remove all "\r" characters, thus it does not matter which newline encoding you use.

We will use a different docs.txt file **during grading**. In particular, it might be longer. The file will never be empty and will only contain ASCII characters.

`Alice.java`

To run Alice, type command:

```
java Alice <BobIP> <BobPort>
```

For example:

```
java Alice sunfire.comp.nus.edu.sg 9000
```

connects to Bob running on **`sunfire.comp.nus.edu.sg`** at port 9000.

Upon startup, Alice will read Bob's RSA public key from the file **`public.key`** stored in the same directory as **`Alice.java`**.

Alice runs a TCP client. When connected to Bob, Alice generates an AES key, encrypts it with Bob's public key and sends it Bob. She then receives multiple encrypted messages from Bob, decrypts and saves them to the file **`msgs.txt`** in the same directory as **`Alice.java`**.

Note that Bob's RSA public and private keys remain unchanged across sessions. However, each time **`Alice.java`** launches, a new AES session key should be generated and sent to Bob.

Furthermore, **`Bob.java`** and **`Alice.java`** should be placed in two different directories.

Useful Java Cryptography Classes

Java **`Cipher`** class provides the functionality of a cryptographic cipher for encryption and decryption. It forms the core of the Java Cryptographic Extension (JCE) framework. A **`Cipher`** object needs to be initialized to **`ENCRYPT_MODE`** or **`DECRYPT_MODE`** before respective operations. Encryption algorithms such as RSA, DES and AES are supported (there is no need to study the feedback mode and padding scheme of these algorithms).

Bob encapsulates (and encrypts) messages as Java **SealedObject**s before transmission. **SealedObject**s are containers which encrypt and decrypt their contents (objects) with the help from a **Cipher** object. The reason we use **SealedObject**s is that it is very easy to transmit objects over TCP. An alternative approach is to transmit byte arrays over TCP (e.g., **doFinal** method of the **Cipher** class actually returns an encrypted message as a byte array). However, we will stick to **SealedObject** in this assignment.

To generate an AES key, Alice should use **KeyGenerator** class to generate an AES **SecretKey** object. Alice also encapsulates AES key as a **SealedObject** before sending it to Bob. However, sealing an AES **SecretKey** object with RSA presents a problem because RSA algorithm imposes a size restriction on the object being encrypted (typically 117 bytes). An AES **SecretKey** object is too large for RSA encryption. The way out is to use RSA to seal the "encoded form" of an AES **Key** object (encoded form can be retrieved using **getEncoded** method of **SecretKey** class). Bob will restore AES **Key** object from the received code.

You may want to read relevant Java API documentation for more information on the aforementioned Java classes. There is no need to use other Java cryptography classes not imported in the given skeleton (using other classes implies you are taking a complex approach).

Plagiarism Warning

You are free to discuss this assignment with your friends. But, ultimately, you should write your own code. We employ zero-tolerance policy against plagiarism. We check systematically for similarities among submissions. If a suspicious case is found, student would be asked to explain his/her code to the evaluator in face. Confirmed breach may result in zero mark for this assignment and further disciplinary action from the school.

Question & Answer

If you have any doubts on this assignment, please post your questions on IVLE forum or consult the teaching team. Teaching staff are not supposed to debug programs for you. However, we may help clarify misconceptions or give you necessary directions.

Disclaimer

This assignment is designed as a proof of concept and is not very secure in the practical sense. To write "more secure" applications, you may want to read the security courses offered by SoC.

You are not allowed to post your solutions in any public domain on the Internet.